Universidad Carlos III de Madrid

Escuela Politécnica Superior

Sistema para la monitorización y estudio de aplicaciones maliciosas para Android

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Autor: Sergio Gutiérrez Hernández

Tutor: Guillermo Suarez de Tangil Rotaeche

Septiembre 2014

Página en blanco.

“In life, unlike chess, the game continues after checkmate”

Isaac Asimov

Índice de Contenidos

Índice de Figuras

Índice de Tablas

Sección 1

RESUMEN

1.1. Resumen

Los teléfonos móviles inteligentes (smartphones) se han posicionado como aparatos altamente populares con grandes capacidades de cómputo, comunicación y sensorización. Estos dispositivos se caracterizan por su habilidad para incorporar aplicaciones desarrolladas por terceras partes, capaces de acceder a los servicios proporcionados por el smartphone. En este contexto, hemos podido observar durante los últimos años un crecimiento exponencial en el número de muestras de aplicaciones haciendo un uso malicioso de los recursos del teléfono [1].

Los recursos necesarios para la detección de comportamiento malicioso son muy elevados, además de ser una tarea que se presenta sumamente compleja. Esta complejidad se ve esencialmente agravada por las grandes restricciones de batería a las que están sometidos estos dispositivos. Por ello, recientes trabajos de investigación han propuesto el uso de la cloud para externalizar las tareas de detección.

Este Trabajo de Fin de Grado aborda el uso de la cloud para aliviar los procesos de análisis de malware. En concreto, presenta la implementación de un sistema que permite mantener una réplica en tiempo real del dispositivo Android de un usuario en la nube.

De manera inicial se realiza un análisis en profundidad del marco actual en el que se encuentra el malware desarrollado para este sistema operativo. También se analizan las tecnologías existentes para hacer frente a estas aplicaciones.

Partiendo de este análisis, el sistema desarrollado detecta las acciones que se realizan en un dispositivo físico y las replica en la nube. De esta forma, el sistema permite usar dicha réplica para realizar estudios avanzados de malware.

El proyecto resultante tiene como finalidad dar soporte a los analistas de malware en el estudio de aplicaciones, proporcionando un entorno controlado para la ejecución de malware que permita la integración de otras herramientas.

1.2. Abstract

Smartphones are rapidly emerging as popular devices with increasingly powerful computation, networking and sensing capabilities. These devices are characterized by their ability to incorporate third party applications capable of accessing the services provided by the smartphone. In this regard, we have experienced a rapid growth in the number of malicious apps during the last few years [1].

Detecting malware is complex task that requires extremely powerful computing resources. This task is aggravated by the huge battery constraints subject to these devices. Thus, recent research has proposed the use of the cloud to outsource detection tasks.

This Bachelor Thesis uses the cloud to facilitate the malware analysis process. Particularly, it presents the implementation of a system that maintains a replica of an Android device in the cloud to better analyze malware.

Initially, we perform an in-depth analysis of current malware detection system. Based on this analysis, we developed a system capable of detecting the actions that take place in a physical device and replicate them into the cloud. This way, the system can use the replica to perform advanced malware analysis tasks.

The resulting project is intended to support security analysts during the analysis of malware, providing them with a controlled environment for the execution of samples. Other malware analysis tools can be used together with this framework.

Página en blanco.

Sección 2

INTRODUCCIÓN

2.1. Motivación

La seguridad en las TIC supone un reto apasionante para cualquier organización. Debe haber una proactividad por parte de los administradores de sistemas (e idealmente también de los usuarios) para mantener los equipos actualizados, libres de vulnerabilidades, y por tanto, seguros frente a cualquier amenaza que pueda materializarse. Por ello, el uso de dispositivos móviles, tanto a nivel personal como profesional debe tener en cuenta la seguridad.

Android es un sistema operativo que apareció por primera vez en 2008 y que ha ido evolucionando en el mercado haciéndose hueco entre otras grandes compañías hasta alcanzar la cuota máxima de venta de dispositivos móviles. El siguiente gráfico muestra la cuota de mercado que han tenido las principales firmas electrónicas que producen smartphones.

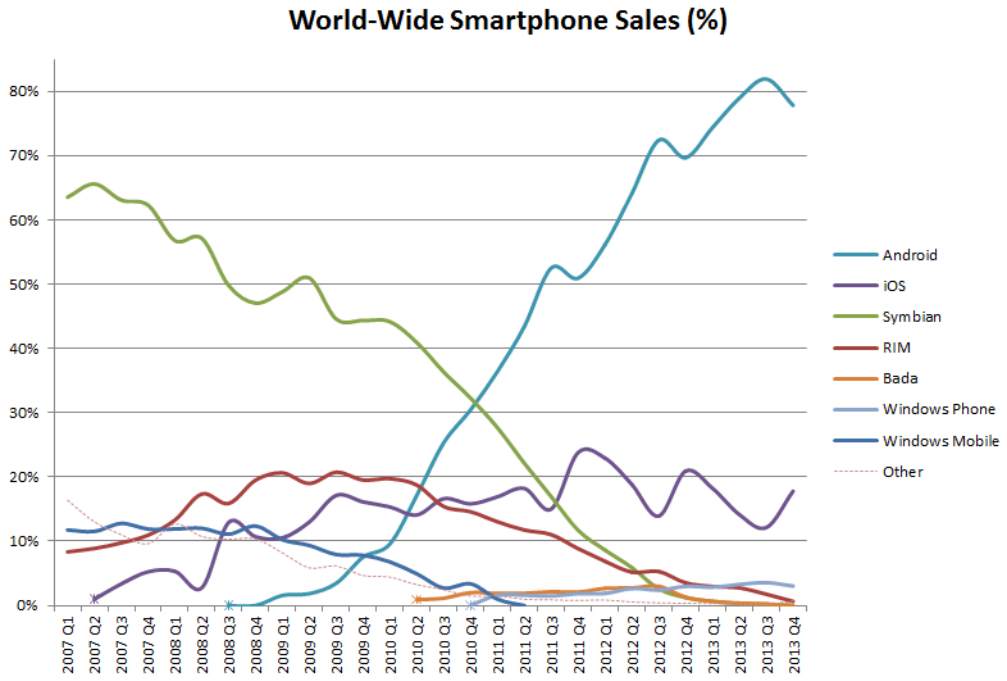


Figura - 1 - World-Wide Smarphone Sales[1]

Durante todo su ciclo de vida este sistema operativo ha supuesto un gran reto en cuanto a seguridad, sus principales herramientas son el uso de permisos y separación lógica de aplicaciones en máquinas virtuales individuales. A pesar de haber orientado su desarrollo hacia un sistema seguro desde el principio, existen diferentes factores que lo hacen susceptible de ser vulnerable. Algunos de estos factores son:

1. Se trata de un sistema muy complejo, con una gran funcionalidad.
2. Como es lógico, los desarrolladores pueden cometer errores de diseño.
3. La seguridad no solo depende de Android, los distribuidores desarrollan el firmware y deciden sobre las actualizaciones de Android.

[1] Fuente: Wikipedia, basado en estudios realizados por Gartner

4. Los usuarios no están formados y no son conscientes de la realidad a la que se enfrentan con el uso de estos dispositivos.

La revolución que ha demostrado este sistema supone un punto de inflexión muy importante para los desarrolladores de malware, que han visto en Android un sistema muy tentador del que lucrarse ilícitamente. El boom de aplicaciones maliciosas que ha habido a lo largo del tiempo se debe principalmente a varias razones:

1. La gran aceptación de los usuarios por este nuevo sistema operativo, y por tanto la gran cantidad de terminales móviles que se venden cada año.
2. La integración de los dispositivos con actividades que requieren de información sensible como acceso a cuentas bancarias, redes personales, etc.
3. El sistema de código abierto y la poca seguridad o medidas que se han establecido para detectar malware en el market oficial.
4. El uso de markets ilícitos por parte de los usuarios, que bien por ignorancia de los mismos o por búsqueda de aplicaciones más económicas han hecho que estos markets estén repletos de malware.

Así el rápido y extenso crecimiento de los teléfonos móviles inteligentes junto con su alta aceptación por parte de los usuarios provoca la aparición de nuevos retos desde el punto de vista de la seguridad de la información. Ya sólo durante el año 2011, el número de aplicaciones Android maliciosas creció en un 3325.5% según un informe publicado por Juniper [2].

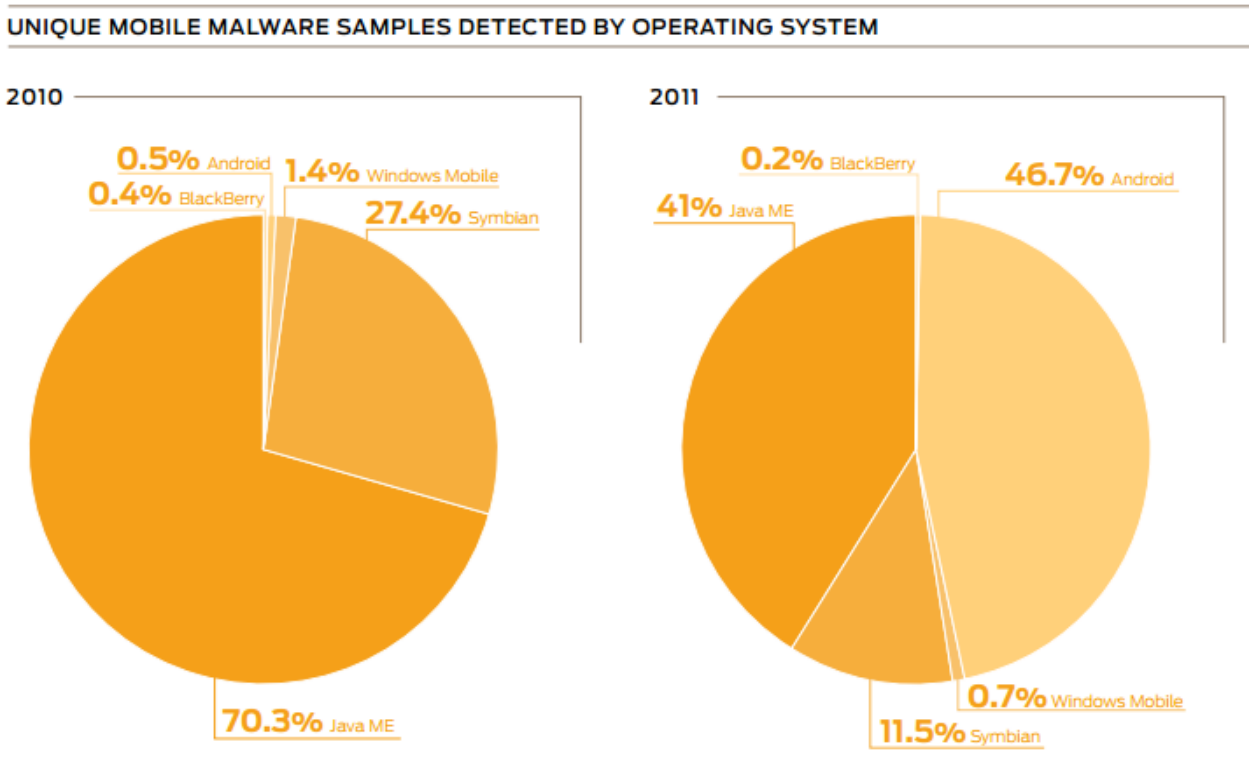


Figura - 2 - Evolución del malware desarrollado para Android entre 2010 y 2011

La evolución [3] que ha tenido el desarrollo de malware para Android desde 2011 hasta 2013 está estrechamente ligada con la cuota de mercado. En 2013 el 92% del malware para aplicaciones móviles está desarrollado para Android.

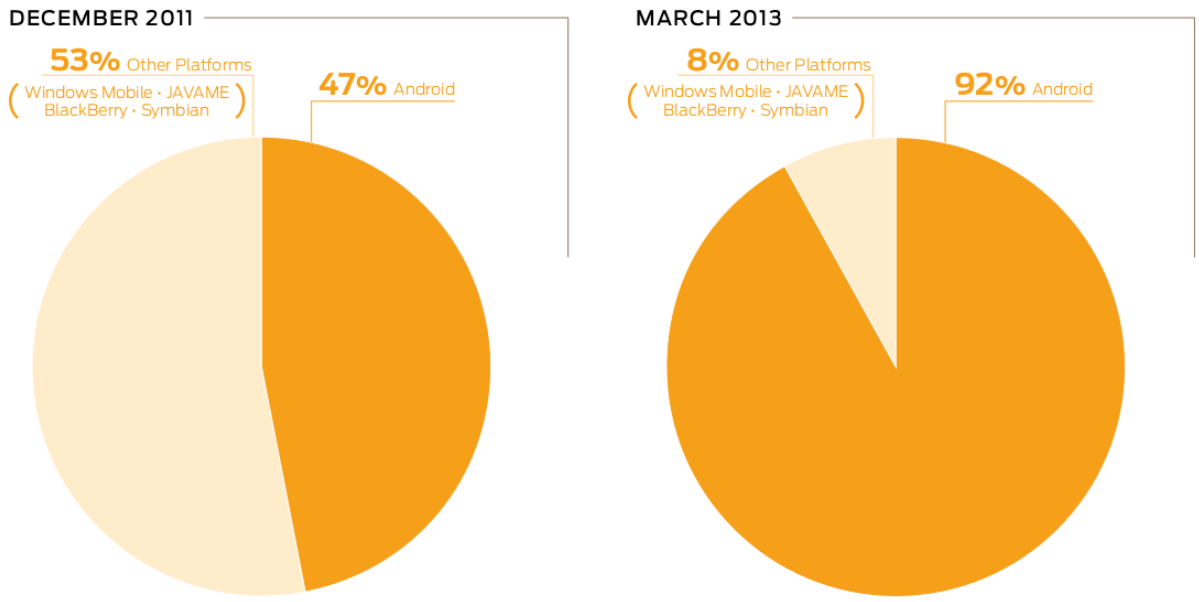


Figura - 3 - Evolución del malware desarrollado para Android frente a otras plataformas

Para hacer frente a este problema, Google ha desarrollado un sistema que de manera automática realiza análisis en busca de malware a las aplicaciones subidas a Play Store. Se trata de Bouncer[2], aprende el comportamiento analizado en unas aplicaciones y lo utiliza para el análisis de otras.

Por otro lado, las principales compañías de antivirus que han desarrollado herramientas de detección, han evolucionado únicamente hacia antivirus instalados como cliente en los propios dispositivos. Estas herramientas consumen gran cantidad de recursos para ser eficaces, lo que supone un grave impacto en la batería del dispositivo, cuya eficiencia está muy limitada hoy en día. Por lo general, estos antivirus no son eficaces frente a malware avanzado, y siempre van un paso por detrás.

El desarrollo del presente trabajo se basa en el uso de la nube para mitigar los problemas de rendimiento causados por el análisis en el terminal físico. La idea de crear un sistema controlado de monitorización de aplicaciones Android surge inicialmente de un trabajo llamado Paranoid Android [4]. Paranoid Android muestra las posibles bases para la creación de un sistema que va más allá del análisis de aplicaciones malware de forma estática. Pretende generar copias exactas de dispositivos móviles en entornos virtuales controlados, donde posteriormente aplicar múltiples técnicas de monitorización de malware.

[2] Android Security Bouncer

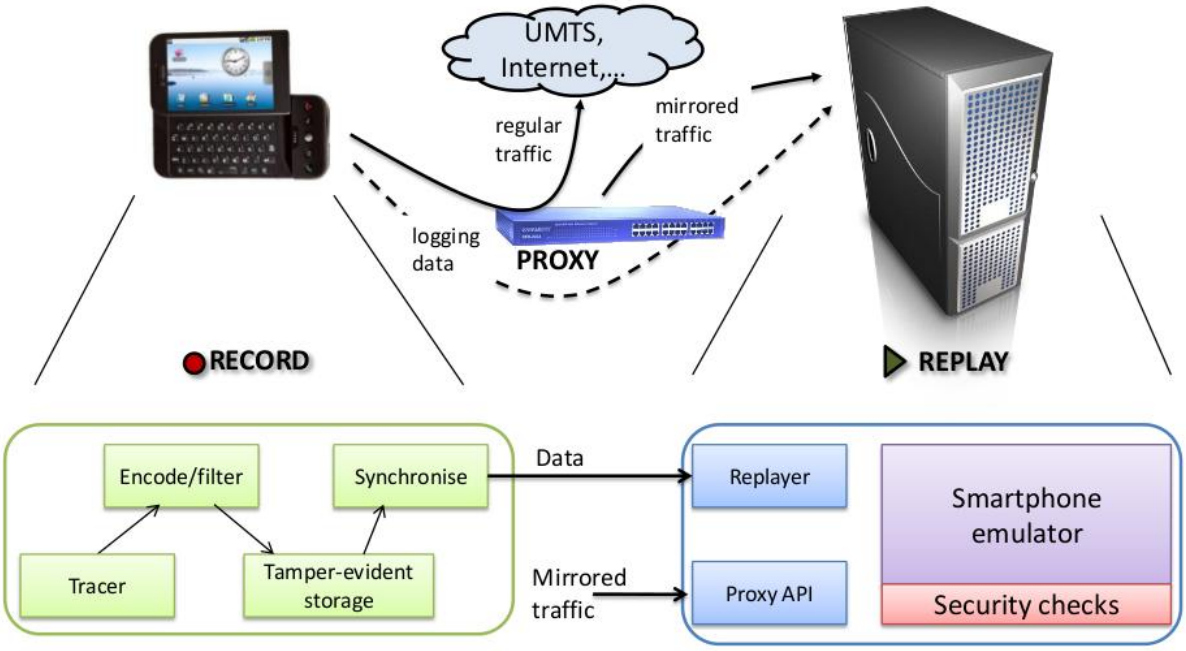


Figura - 4 - Paranoid Android Architecture

De esta manera se posibilita realizar de manera eficiente análisis dinámico sobre la cloud. El análisis dinámico se basa en el desarrollo de las acciones y el comportamiento a lo largo del tiempo de un sistema, por lo que va más allá de un análisis estático.

Actualmente los principales antivirus realizan únicamente análisis estáticos de detección de firmas y técnicas de ingeniería inversa. Los desarrolladores de código malware pueden fácilmente tomar precauciones para que sus aplicaciones pasen inadvertidas frente a estos análisis estáticos. Las principales técnicas que utilizan son ofuscación de código y reordenación del mismo con operaciones XOR.

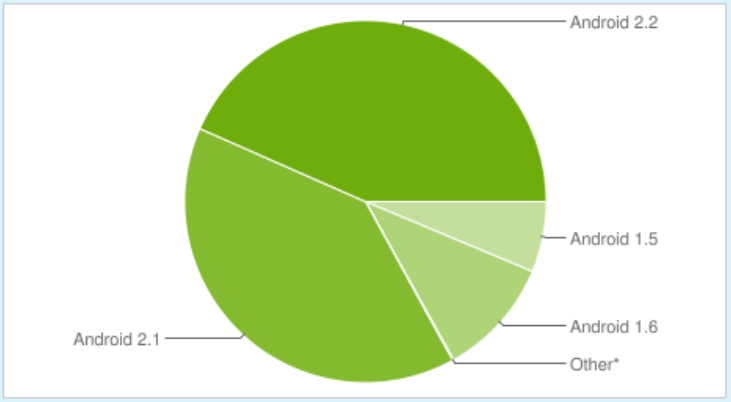
Las ventajas del sistema que se va a implementar, y por tanto de ejecutar malware en un entorno controlado, es: poder almacenar la secuencia de acciones que realiza la aplicación, para replicarla cuantas veces se desee y analizar de manera dinámica en la cloud el comportamiento de la misma, comparar trazas generadas por diversos malware para establecer relaciones de familias y semejanza entre distintos tipos de aplicaciones.

2.2. Objetivos

El presente Trabajo de Fin de Grado tiene como objetivo principal la creación de un sistema que replique en tiempo real el comportamiento de un dispositivo Android a la nube. El comportamiento que se monitorizará y replicará serán acciones básicas de usuario, así como los principales eventos que una aplicación malware puede generar. La nube donde se replicará el comportamiento es un entorno controlado o sandbox, formado por un Dispositivo Virtual Android donde se puedan aplicar técnicas dinámicas de análisis malware. El sistema estará basado en la integración y mejora de distintas herramientas de código libre que existen actualmente para el análisis y monitorización de aplicaciones malware.

Para satisfacer el dinamismo y mejorar la flexibilidad del sistema frente a futuros desarrollos por la comunidad, la herramienta será independiente de la versión de Android que implementen los dispositivos involucrados.

Cada nueva versión de Android implementan mejoras gráficas, de sistema, seguridad, novedades hardware, etc. Supone una renovación y un avance en Android, pero al mismo tiempo hace que las versiones antiguas se queden obsoletas, y por tanto que dejen de utilizarse.



Platform	API Level	Distribution
Android 1.5	3	6.3%
Android 1.6	4	10.6%
Android 2.1	7	39.6%
Android 2.2	8	43.4%

Data collected during two weeks ending on December 1, 2010

** Other: 0.1% of devices running obsolete versions*

Figura - 5 - Distribución de Terminales por API en 2010

Como se observa, durante los comienzos de expansión de Android en el 2010 la versión que más terminales tenía es la 2.2 (Froyo).

A mediados de 2014 la última versión en salir al mercado es la 4.4 (KitKat). La distribución[3] de versiones que hay más reciente es:

[3] Distribución de terminales Android por API

Version	Codename	API	Distribution
2.2	Froyo	8	0.7%
2.3.3 - 2.3.7	Gingerbread	10	13.6%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	10.6%
4.1.x	Jelly Bean	16	26.5%
4.2.x		17	19.8%
4.3		18	7.9%
4.4	KitKat	19	20.9%

Data collected during a 7-day period ending on August 12, 2014.
Any versions with less than 0.1% distribution are not shown.

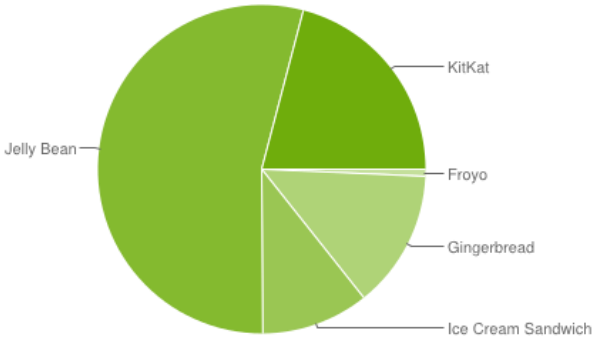


Figura - 6 - Distribución de terminales Android por API en Agosto 2014

Con el conocimiento inicial que se tiene de Android y de los métodos de conexión a dispositivos físicos para obtener información, surge otro objetivo relevante. Además de que la replicación del comportamiento se realice a través de USB, se priorizará que el proyecto sea transparente al tipo de conexión utilizada. Se implementarán medidas que permitan la replicación en la nube a través de redes inalámbricas. Esto supone romper de lleno con las limitaciones que existen al tener el dispositivo conectado por cable al sistema. Por ejemplo permitiría la movilidad del dispositivo y por tanto la obtención de la localización según el intervalo de tiempo que se estime necesario.

Una vez realizado un sistema que sirva como base para la replicación de acciones, se considerará si forma parte del alcance la integración de sistemas de monitorización de aplicaciones malware. Este sistema sería el punto de partida para hacerse un hueco en la comunidad de desarrolladores de herramientas orientadas al análisis malware en Android. Por ello debe ser dinámico, flexible y que permita la integración de proyectos ya existentes generando nuevas ideas o líneas de investigación.

2.3. Estructura del documento

- **Resumen**

De manera breve se explica cuál es el problema que se aborda y la solución planteada en este Trabajo de Fin de Grado.

- **Introducción**

Se define la motivación y los objetivos del sistema a desarrollar.

- **Estado del Arte**

Para poder definir el sistema es necesario realizar este estudio previo sobre el malware y las herramientas de detección de aplicaciones maliciosas y el estado actual de la replicación en la nube.

- **Estudio de viabilidad**

Consiste en la definición de los requisitos a alto nivel y el alcance del sistema basado en el estudio del estado del arte. Se realiza la gestión del proyecto y propone además un calendario de trabajo.

- **Análisis**

Se definen los requisitos que debe satisfacer el sistema, y se analiza en profundidad qué eventos deben poder ser replicados en la nube.

- **Diseño**

Se traslada el análisis a un nivel más bajo, definiendo los módulos del sistema y la integración entre ellos.

- **Implementación**

A un nivel de programación se detallan los módulos y las clases que lo componen.

- **Evaluación e Implantación**

Se realizan las pruebas en base a los requisitos identificados y se definen los pasos para implantar el sistema.

- **Conclusiones**

Para finalizar se obtienen conclusiones del sistema realizado y los resultados del mismo.

- **Anexos**

Se adjuntan las tablas que detallan los eventos analizados, el diagrama de gantt inicial así como la replanificación sufrida y por último la bibliografía y referencias utilizadas.

2.4. Glosario

- **BotNet:** Conjunto de programas interconectados que se comunican unos para realizar tareas similares o de manera conjunta.
- **C&C server** (Command and Control Server): Son servidores capaces de enviar comandos para ejecutar a miembros que forman parte de su botnet.
- **Cloud:** Paradigma que ofrece servicios de computo a través de internet.
- **Exploit:** Sistema que aprovecha una vulnerabilidad para realizar acciones o ganar accesos de forma no autorizada.
- **Freelancer**: Trabajador autónomo o independiente.
- **Sandbox**: Entorno de pruebas que permite ejecutar software evitando cambios en los sistemas.

2.5. Acrónimos

- **API:** Application Programming Interface.
- **AVD**: Android Virtual Device.
- **CCN**: Centro Criptológico Nacional.
- **IOS**: Iphone Operative System.
- **LOPD**: Ley Orgánica de Protección de Datos.
- **SDK**: Software Development Kit.
- **TFG**: Trabajo Fin de Grado.
- **TIC**: Tecnologías de la Información y las Comunicaciones.
- **GPS**: Global Positioning System.
- **UE**: Union Europea.
- **XOR**: Exclusive Or.

Sección 3

ESTADO DEL ARTE

3.1. Android

Android fue ideado por la empresa Android Inc. en el año 2003. Consiste en un Sistema Operativo basado en el Kernel de Linux para dispositivos móviles. Rápidamente Google se interesó por este proyecto y en 2005 adquirió Android Inc. para dos años más tarde, en 2007 dar a conocer el sistema bajo el nombre de Android Open Source Project. El encargado de desarrollar este proyecto es un consorcio conocido como The Open Handset Alliance. Este consorcio se compuso inicialmente por 78 sociedades distintas para dedicarse al desarrollo de estándares para dispositivos móviles. El primer terminal móvil con sistema operativo Android apareció en 2008 bajo el nombre de HTC Dream.

Este Sistema Operativo está ideado, y por tanto optimizado, para dispositivos portátiles que funcionan con baterías de baja potencia, pero que a su vez contienen gran cantidad de hardware como por ejemplo: GPS, cámara de fotos, sensores de luz u orientación, WiFi, servicio de datos móviles, pantalla táctil, etc. Al igual que otros sistemas para terminales móviles Android permite a las aplicaciones utilizar dichas herramientas hardware a través de la abstracción y proporciona un entorno definido para las aplicaciones de usuario.

Se trata de un sistema de código abierto y Google permite su desarrollo y modificación bajo licencia Apache. Esto implica que distintas versiones de Android pueden ser distribuidas libremente por los fabricantes de dispositivos, proveedores de servicios inalámbricos y desarrolladores entusiastas.

El diagrama de arquitectura de Android es el siguiente:

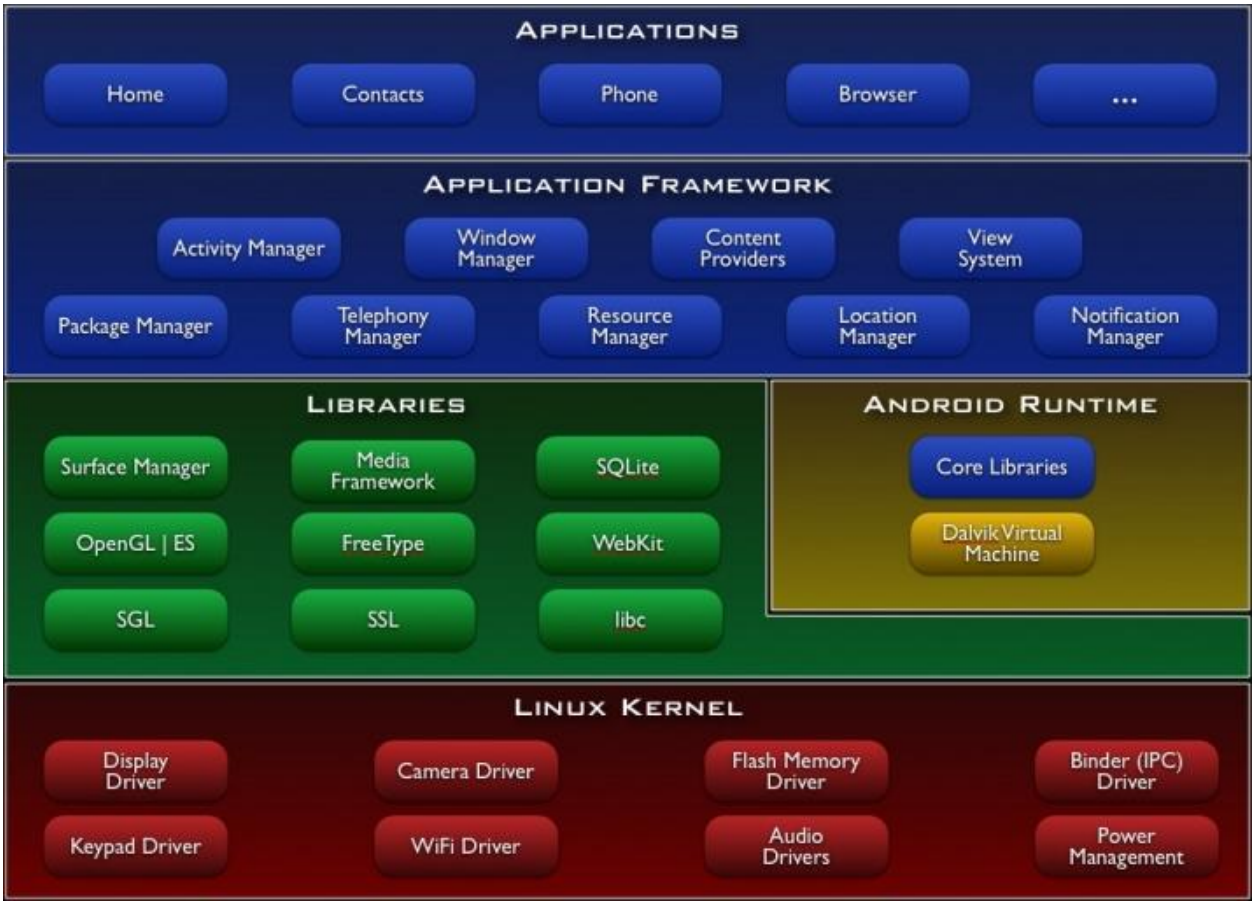


Figura - 7 - Android System Architecture[4]

Como se observa en el diagrama, Android se subdivide en cinco capas: en el nivel más bajo el kernel y el hardware, por encima las librerías nativas y el sistema de ejecución Android Runtime, después el framework de aplicaciones y en la parte superior del todo las aplicaciones de sistema. Las librerías están desarrolladas en C/C++, mientras que el framework de aplicación lo está en Java. Las aplicaciones se ejecutan en la máquina virtual Dalvik, que a efectos prácticos de comparación con Java es su homólogo en Android.

- El núcleo usado es Linux 2.6, modificado para las necesidades especiales de gestión de energía, gestión de memoria y el entorno de ejecución. Esta capa proporciona el nivel de abstracción entre los elementos hardware y las capas superiores. Por encima del kernel se ejecutan algunos demonios típicos de Linux como el de Bluetooth o el WPA_Supplicant para el cifrado WiFi.
- Como Android debe funcionar en dispositivos con poca memoria y con procesadores de baja potencia, las bibliotecas son nativas de Android (C++) y se han optimizado para tal fin. Esta categoría incluye además las bibliotecas basadas en Java que son específicas para el desarrollo de Android. Los ejemplos de las bibliotecas en esta categoría incluyen las

[4] Android System Architecture

correspondientes al marco de aplicación, además de las que facilitan la construcción de interfaz de usuario, dibujo de gráficos y acceso a la base de datos.

- El componente Android Runtime consiste en la máquina virtual Dalvik y en las propias librerías que la manejan. La máquina virtual Dalvik es un intérprete de código de bytes, archivos .dex, que son una transformación muy eficiente de los archivos .class de Java. Con ello se consigue ejecutar código muy pesado en Java en dispositivos con una gran limitación de memoria, bien sean smartphones, tablets o dispositivos empotrados,
- El framework de aplicación es un conjunto de librerías (APIs) desarrolladas en Java que conforman el entorno en el que se ejecutan y gestionan las aplicaciones de Android. Este marco implementa el concepto de que las aplicaciones de Android se construyen a partir de componentes reutilizables, intercambiables y reemplazables, por lo que los usuarios pueden sobrescribir métodos implementados si lo consideran necesario en su desarrollo.
- Finalmente la capa de aplicaciones que permiten la interacción del usuario con el terminal. Estas comprenden tanto las aplicaciones nativas proporcionadas con el sistema operativo (aplicaciones de sistema como por ejemplo, navegador web, gestión de configuración, aplicaciones propias de cada fabricante, etc.) como las aplicaciones de terceros instaladas por el usuario. Las aplicaciones se ejecutan en su propio espacio en la máquina virtual de Dalvik y puede estar formado por diversos componentes: actividades, servicios, *Broadcast*, content providers, etc.

Android, al ser código libre, cuenta con una gran comunidad de desarrolladores (tanto de aplicaciones como de firmwares) que está en constante evolución y permiten ampliar y mejorar el sistema a través de sugerencias y detectar en poco tiempo vulnerabilidades para puedan ser subsanadas.

El desarrollo de las aplicaciones Android es en lenguaje Java que utiliza las librerías propias de Android, conocidas como APIs. A continuación se muestra el histórico de APIs proporcionadas por la SDK de:

API 3 - Android 1.5 - Cupcake	**API14 - Android 4.0 - Ice Cream Sandwich**
API 4 - Android 1.6 - Donut	**API15 - Android 4.0.3 - Ice Cream Sandwich**
API 7 - Android 2.1- Eclair	**API 16 - Android 4.1.2 - Jelly Bean**
API 8 - Android 2.2 - Froyo	**API 17 - Android 4.2.2 - Jelly Bean**
API 10 - Android 2.3.3 - Gingerbread	**API 18 - Android 4.3 - Jelly Bean**
API 11 - Android 3.0 - Honeycomb	**API 19 - Android 4.4.2 - KitKat**
API 12 - Android 3.1 - Honeycomb	**API 20 - Android 4.4W - KitKat Wearable**
API 13 - Android 3.2 - Honeycomb	

Tabla - 1 - APIs de Android

Actualmente el desarrollo de aplicaciones Android permite la inclusión de contenido web que se muestra como si fuese código nativo (a través del motor de renderizado WebKit). Esto ha permitido la evolución de aplicaciones pudiendo ser desarrolladas en lenguajes web como HTML5, estilos CSS y Javascript que a través del código nativo que implementa el dispositivo, permiten la generación de aplicaciones híbridas.

3.2. Evolución del Malware en Android

A continuación se expone de manera breve la historia del malware en Android considerando las etapas más relevantes y los ejemplos más significativos. Este estudio está basado en reportes de distintos organismos que se dedican a la seguridad informática {[14], [15], [16], [17], [18]} ya que no hay un mecanismo estandarizado que asegure la veracidad de los datos numéricos al cien por cien.

3.2.1. Etapa 2010

El malware en Android surgió a mediados de 2010, con la primera aplicación conocida como FakePlayer y perteneciente a la familia *ANDROIDOS_DROIDSMS.A*, siendo esta aplicación en particular un Trojano. Se hace pasar por un reproductor de medios, pero una vez instalado la aplicación comienza a enviar SMS's a números premium, conllevando un beneficio para el propietario del número, y un gran gasto para el usuario del terminal. Esta aplicación no es la única de esta familia, ya que enseguida aparecen otras como *SMS.AndroidOS.FakePlayer.c* y *SMS.AndroidOS.FakePlayer.b*, las cuales supuestamente proporcionan contenido sexual.

El resto del año 2010 surgen las primeras familias de malware, que acceden a datos como las coordenadas GPS, IMEI del dispositivo, etc. Además aparece la primera BotNet que afecta a Android, como es el caso de la aplicación *Android.Gemini*, que recibe y permite ejecutar comandos de servidores extranjeros (principalmente de China y Rusia), permitiendo controlar el dispositivo de manera remota. Esta aplicación es un reempaquetado, que consiste en tomar como base una aplicación ya existente y legítima (un juego por ejemplo) donde se cambia parte del código para añadir la actividad maliciosa.

3.2.1. Etapa 2011

Enseguida los desarrolladores se dan cuenta del gran beneficio económico que supone lucrarse a través de los dispositivos móviles. El primer boom de malware tuvo lugar a finales del año 2011, donde se desarrollan muchas variantes del mismo tipo, además de comenzar a surgir nuevas familias que se dedican a realizar otro tipo de actividad maliciosa. Sólo en Septiembre de 2011, el número de malware nuevo se incrementó en más de un 30%. A partir de aquí existe un cambio muy importante en el comportamiento malicioso, ya que en Octubre de 2011 el porcentaje de aplicaciones que tratan de robar datos personales ascendió un 34%.

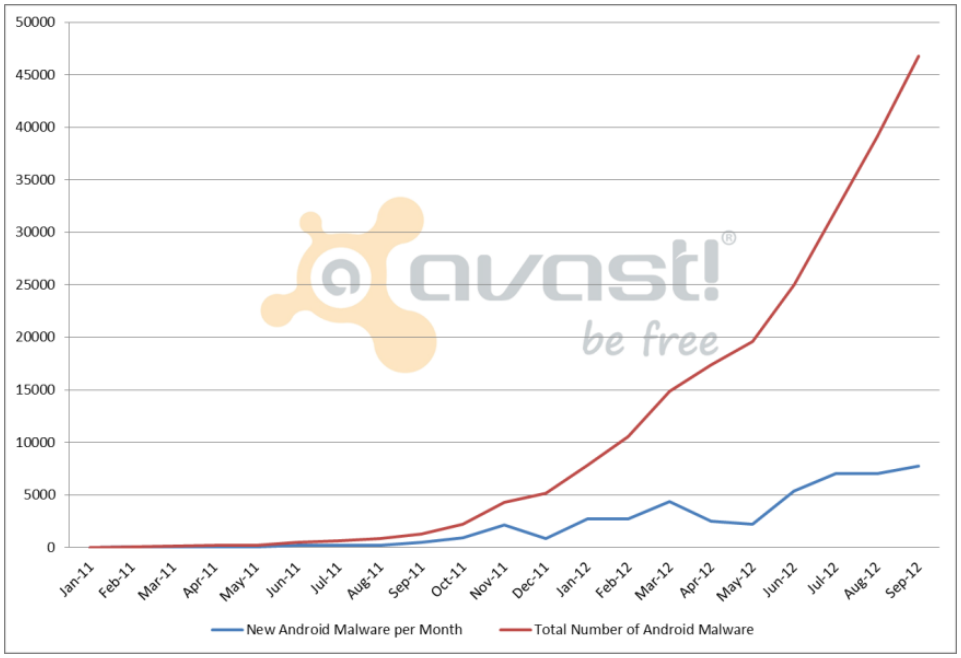


Figura - 8 - Aumento de malware en Android periodo 2011 - 2012

La mayoría de este nuevo software se corresponde con las familias DroidKungFu y AnserverBot, además de todas sus variantes. En particular, *Android/DroidKungFu.A* utiliza el mismo exploit que DroidDream para obtener permisos de administrador e instalar el componente principal del malware, capaz de ejecutar comandos, descargar una APK específica de una URL específica, además de iniciar la instalación de esa APK. Además obtiene información como el IMEI, operador de red y números de teléfono de los contactos. Por lo tanto esta aplicación se trata de una BotNet.

Otro ejemplo muy relevante por la complejidad de su implementación es el malware AnserverBot. Consiste en una aplicación reempaquetada a la cual se le han añadido dos aplicaciones ocultas: *anservera.db* y *anserverb.db*. Ambas tienen funcionalidad diferente, pero trabajan en conjunto. Una vez ejecutada la aplicación host se lanza un mensaje de actualización donde se instala *anservera.db*, que corre en segundo plano para ejecutar *anserverb.db* sin necesidad de ser instalada. Esta segunda parte es la que se comunica con el exterior ejecutando el código solicitado por el C&C server. Se trata de una BotNet muy robusta, ya que el sistema funciona incluso si se desinstala la aplicación host, además de ser capaz de actualizarse a sí misma.

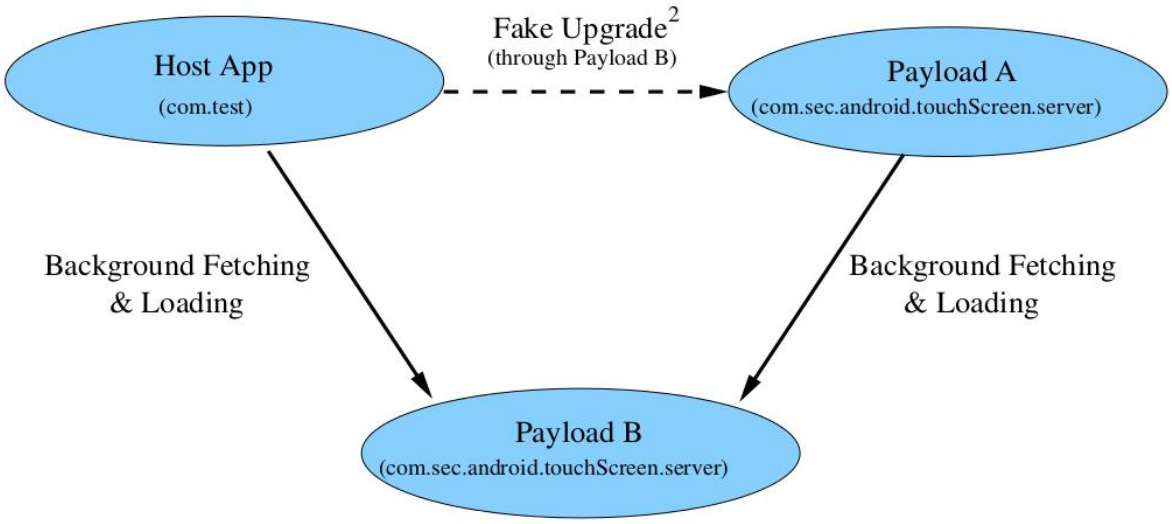


Figura - 9 - Funcionamiento de BotNet AnserverBot

3.2.2. Etapa 2012

A partir de 2012 el desarrollo de malware se puede considerar de nivel exponencial. Nuevas familias y variantes aparecen continuamente debido al afán lucrativo de sus autores.

En la primera etapa de 2012 se cuadruplica el número de familias existentes con respecto al mismo periodo (primer trimestre) de 2011, tomando como ejemplo los datos analizados por F-Secure Labs, la cantidad de instancias malware aumentan de 139 a 3063.

De la misma manera que se comienza a prevenir y detectar las aplicaciones maliciosas por parte de compañías de antivirus y de investigación, los desarrolladores de malware adoptan técnicas de cifrado y ocultación para evitar ser reconocidas por la detección de firmas de antivirus.

Un ejemplo de malware de esta época es el hoy clásico troyano *FakeToken*. Se trata de una aplicación que supuestamente pertenece a una entidad bancaria y genera tokens para acceder al banco web del usuario indicando la clave de operaciones. Esta clave es leída y enviada a un servidor.

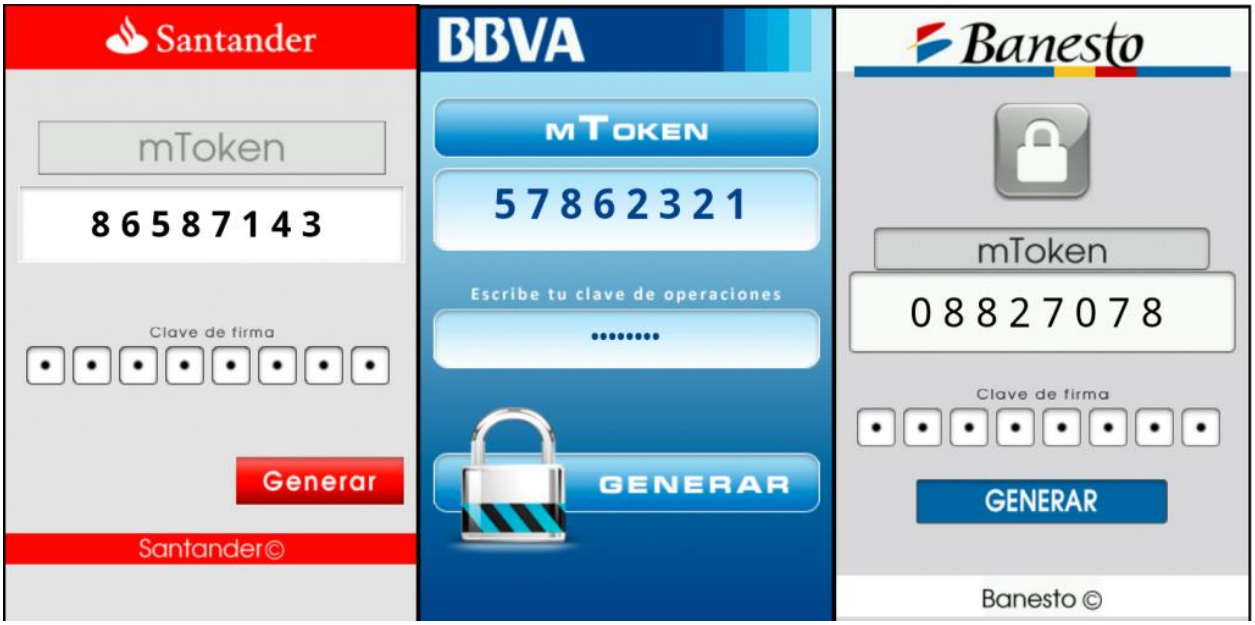


Figura - 10 - Capturas del malware Android *FakeToken*

Un dato muy curioso de mediados de año, es que tan solo en Agosto se detectaron 10 veces más aplicaciones nuevas que en los meses anteriores. Aunque muchas de estas aplicaciones no son más que reempaquetados de malware anterior, donde se ha modificado tan solo el método de ocultación para evitar ser detectado.

A finales de año aparecen nuevas versiones del tipo PremiumSMS ya que dan de alta al usuario a servicios de subscripción de SMS, cobrando por SMS recibido en vez de enviado. La misma aplicación se encarga de detectar y eliminar el SMS recibido antes de que se entere el usuario.

Un ejemplo de aplicación de Diciembre es *Android/AirPush* y sus variantes. Consiste básicamente en lanzar una gran cantidad de notificaciones y crear marcadores en el navegador sobre anuncios y alertas que llevan al usuario a enviar SMS Premium a servidores. Además pueden dar publicidad a mercados ilegales o directamente enlazan con aplicaciones propias (también malware) en este tipo de mercados.

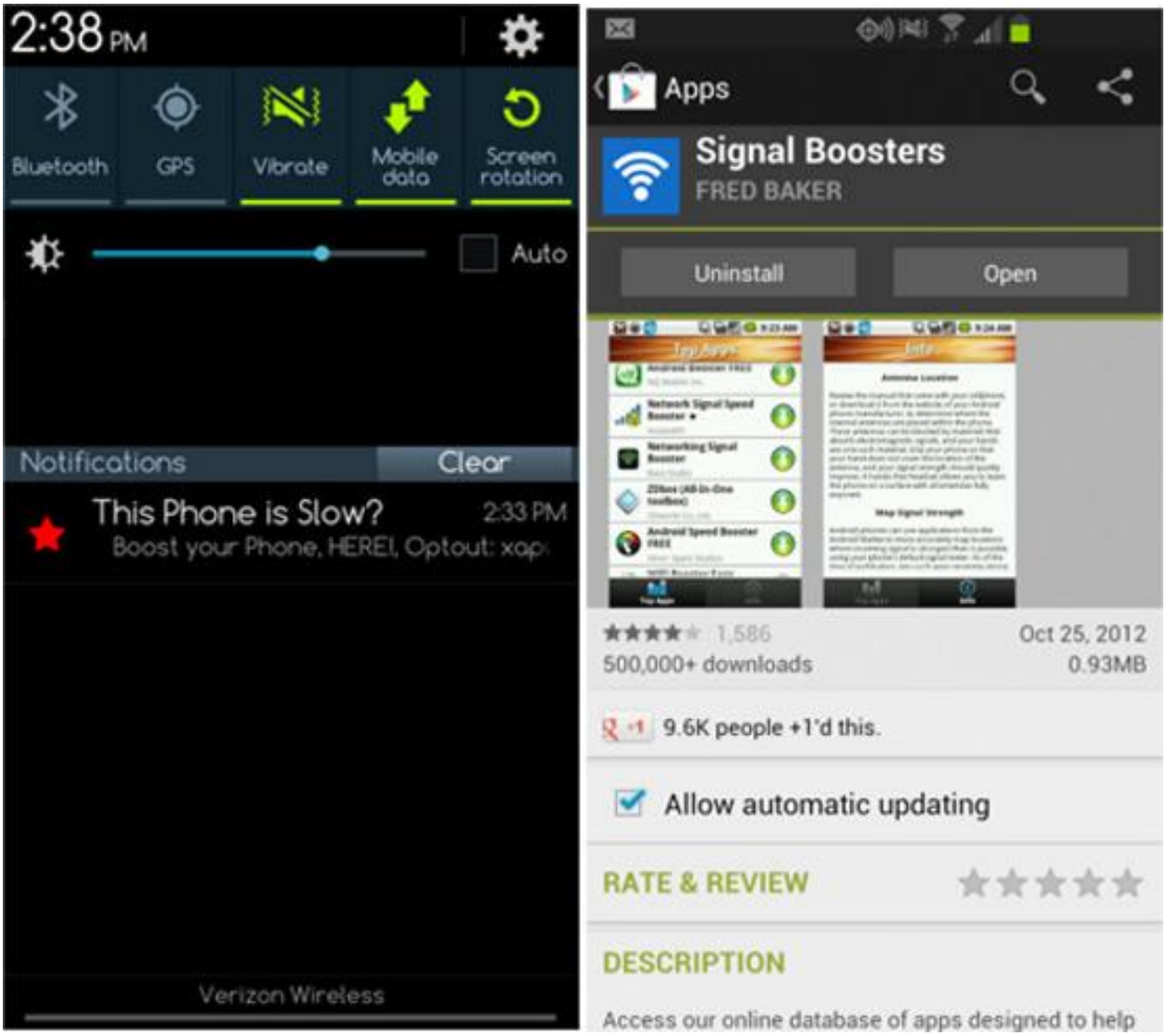


Figura - 11 - Captura de malware para Android *AirPush*

El año 2012 se caracteriza no solo por un gran aumento del malware sino también por la especialización, tanto en nuevas actividades maliciosas como por la capacidad para ocultarse frente a métodos de detección.

3.2.3. Etapa 2013

Los datos que publicó Juniper en su último report del año 2013 sobre amenazas móviles son muy alarmantes. El crecimiento de malware para todas las tecnologías ha aumentado desde Marzo de 2012 a Marzo de 2013 un 614%.

Casi la mitad de todo el malware (48%) pertenece a la familia de TrojanSMS, ya que es el método más lucrativo que existe, al ganar dinero haciendo que tan solo el usuario pulse un botón obligándolo a enviar un SMS premium.

3.2.4. Tipos de Malware

Hacer una buena categorización del malware de Android puede ser muy difícil y ambiguo, ya que no hay claros diferenciadores entre unos y otros, al tener muchos tipos diferentes de aplicaciones características en común. Juniper ha realizado una categorización [3] muy genérica de tres tipos, basándose en todas las aplicaciones existentes entre Marzo de 2012 y Marzo de 2013, ya que estos tres tipos representan 9 de cada 10 aplicaciones de ese conjunto. El tipo mayoritario es el troyano que envía SMS's Premium ocupando el 48% del total. Le sigue el tipo FakeInstaller con un 29%, que solicita permiso al usuario para descargar/actualizar la aplicación, y cuando el usuario acepta se instala la aplicación malware. El último tipo de malware son las aplicaciones espía, que recolectan diversa información y la envían al exterior.

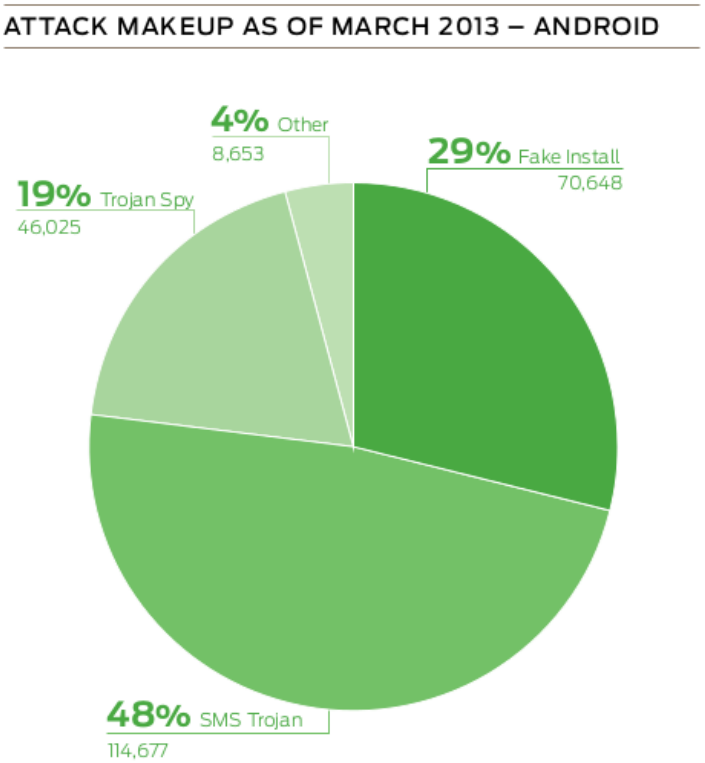


Figura - 12 - Distribución de tipos de malware en Android

De esta gráfica se deduce que lo más importante para los atacantes es obtener un rápido beneficio económico con el envío de SMS, quedando en segundo plano el espionaje de información.

En tan solo un año apenas, con la evolución de nuevas versiones y las redes sociales el malware ha evolucionado creando nuevas familias que se reparten de manera muy equitativa la distribución de la totalidad del código malicioso en Android. El informe [5] de amenazas móviles de Trend Micro para el primer trimestre del año 2014, indica que las familias de malware más representativas en 2013 siguen estando en la cima de la distribución, pese a que aparecen nuevas familias basadas en el robo de información personal y adware.

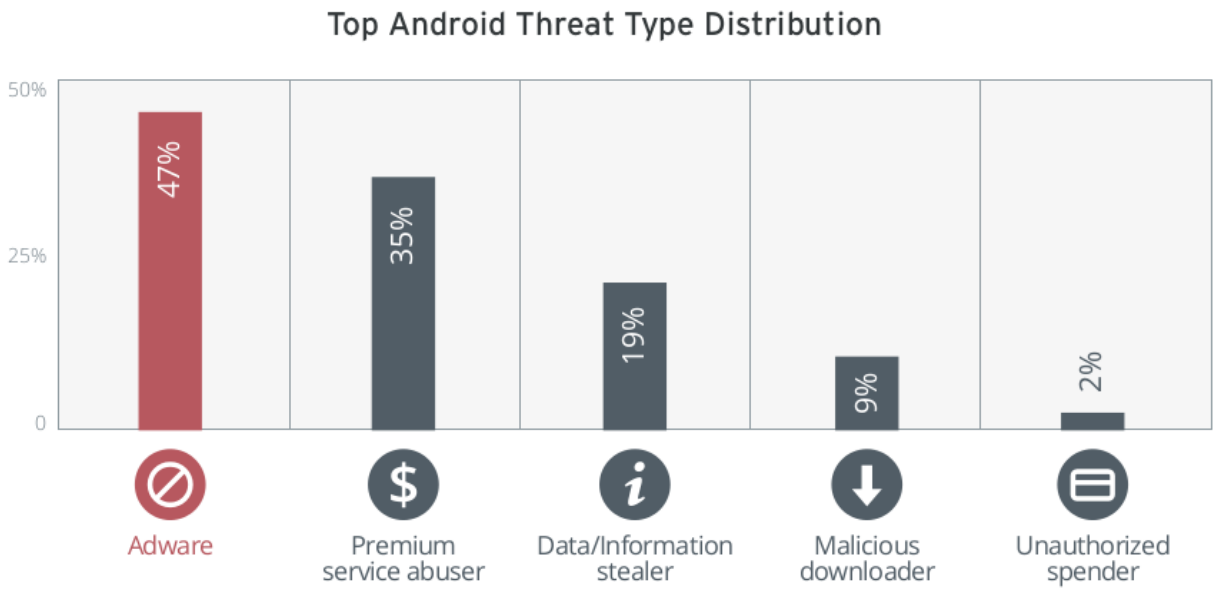


Figura - 13 - Distribución de tipos de malware de Android en #1 trimestre de 2014 [5]

Como ejemplo de las familias más importantes pertenecientes a esta tipología destacan:

- ***OPFAKE***. Troyano que envía SMS Premium y se propaga a contactos.
- ***SMSREG***. Se replica de manera autónoma infectando archivos, programas y enviándose a otros contactos. Elimina archivos y roba información personal.
- ***GINMASTER***. Troyano que roba información y la envía a servidores externos.
- MSEG.Data/Information stealer, seguimiento de pulsado de teclas. Envío de SMS Premium.
- ***FAKEINST***. Se hace pasar por instalador de otras aplicaciones pero envía SMS Premium.
- ***MTK***. Data/Information stealer y Malicious downloader.
- ***SMSPAY***. Adware que envía *spam* al usuario accesos a direcciones con más contenido malware y roba información sensible.
- ***STEALER***. Es similar a MTK.

3.3. Seguridad en Dispositivos Android

Hoy en día no existe una ley o normativa que regule la producción de dispositivos móviles en cumplimiento de unos controles mínimos que se deban seguir para garantizar la seguridad de los mismos. De igual modo, tampoco existe un control exhaustivo sobre los terceros que desarrollan aplicaciones para el Mercado de Google, a pesar de que hay activo un sistema automático de detección de malware denominado Google Bouncer [19] [20], resulta relativamente fácil elaborar malware que no sea detectado por este sistema.

Adicionalmente existe un sistema de aviso o reporte de aplicaciones con contenido problemático para informar a Google de un mal uso de una aplicación. Este sistema puede ser utilizado por los usuarios en conjunto si se detecta el funcionamiento de una aplicación sospechosa. Por norma general, una aplicación que realiza una tarea determinada, por ejemplo un juego simple, un

editor de fotografías, o un lector de *ebooks*, no requiere de más permisos que los necesarios para conectarse a internet, acceder al almacenamiento externo y poco más.

Llama la atención cuando una aplicación tan simple como una grabadora de video, tiene permisos para recibir SMSs, enviarlos, acceder a las cuentas del dispositivo, etc. Existe un caso[5] reciente que ha analizado Avast y que por supuesto, se ha llegado a la conclusión de que se trata de una aplicación maliciosa. De cara al público vende una finalidad de grabación de video, mientras que en segundo plano suscribe a los contactos de aplicaciones de mensajería (Whatsapp o Telegram) a servicios *premium* de SMS.

De manera bastante objetiva estos casos no atienden a lógica alguna, es incomprensible como una aplicación de este estilo, que ni si quiera tiene el código ofuscado, está subida libremente al market de Google sin haber sido identificada como malware.

A diferencia de Google, IOS tiene un sistema [6] de verificación, por el que pasan todas las aplicaciones. Para subir una aplicación de IOS se requiere de una aprobación que generalmente tarda una semana, donde un técnico analiza el código de la misma en busca de malware, y emite una notificación al desarrollador en caso de que se consienta la publicación. En cualquier caso, los usuarios deberían ser más cuidadosos y preguntarse por qué son necesarios los permisos solicitados, y siempre sospechar cuando el desarrollador no es lo suficientemente transparente en la justificación de los permisos.

Por otro lado, existe un gran vacío legal en el cumplimiento por parte de los desarrolladores de una ley de tanto peso como es la Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de carácter personal [7] (LOPD). En concreto, las empresas de software que residan fuera de la UE se atienen a la legislación de su país, por lo que a pesar de desarrollar aplicaciones que pueden adquirir información personal de los dispositivos, no les es aplicable la LOPD.

En España es únicamente aplicable con carácter informativo los manuales publicados por el CCN sobre seguridad en dispositivos móviles [8], así como particularmente en Android [9].

3.4. Antivirus en Smartphones

Como medida preventiva, al igual que en PC's o equipos de escritorio, los antivirus para smartphones han cobrado gran importancia a medida que el malware ha ido evolucionando. Las grandes compañías de seguridad que desarrollan antivirus han ampliado su negocio a los dispositivos móviles, independientemente de la plataforma, ya que existe malware para cualquier sistema operativo.

Los ejemplos más significativos de antivirus para Android son:

- AVG AntiVirus Security
- Avast! Mobile Security & Antivirus
- 360 Security Antivirus
- CM Security AppLock & AntiVirus
- Norton Mobile Security
- Lookout Mobile Security

[5] Nuevo SmsTrojan

- Kaspersky Internet Security for Android
- Mobile Security & Antivirus - TrendMicro

De manera adicional al análisis del sistema, es decir, a las aplicaciones que tiene instalado el dispositivo, los antivirus para dispositivos móviles realizan una gran cantidad de funciones que mejoran la seguridad dando confianza a los usuarios. Entre estas otras funcionalidades podemos encontrar:

- Bloqueo de llamadas/SMS tipo aplicación BlackList
- Monitorización del tráfico de red
- Sistema de backups de archivos en la nube
- Administrador de procesos
- Análisis en tiempo real
- Sistemas antirrobo por localización y bloqueo de terminal
- Formateo de dispositivo

Entrando en el tema que atañe a la detección propia de aplicaciones malware, en general el funcionamiento de todos los antivirus está basado en el análisis de firmas de malware y la actualización constante de las bases de datos de firmas. Por ello, los análisis se realizan de manera estática, quedando limitado en gran medida a lo que se conoce y existe de manera previa, no al comportamiento que pueda ser detectado con un análisis dinámico.

Para probar la gran limitación que tiene el análisis basado en firmas en malware de Android, ya se han realizado diversos estudios que demuestran con transparencia la poca fiabilidad de estos antivirus.

El primer ejemplo [10] viene dado por el Instituto Fraunhofer para la Integración y Aplicación de la Seguridad. Se realizan varias pruebas de infección malware contra dispositivos que poseen antivirus, entre las que destacan la ejecución de malware modificado para evitar el análisis de firmas o modificación de exploits como Exploid, GingerBreak y RageAgainsTheCage que son incluidos como parte de la aplicación maliciosa. Los resultados determinan que únicamente las aplicaciones malware sin modificar son detectadas por los antivirus, mientras que aplicaciones o exploits alterados en pocas ocasiones se detectan correctamente.

El segundo estudio [11] es realizado en la Universidad de Carolina del Norte y demuestran sin lugar a dudas la poca flexibilidad y resiliencia que prácticamente todos los antivirus considerados de calidad en Android presentan frente a técnicas de evasión y ofuscación del código que los desarrolladores de malware suelen utilizar. Los investigadores de esta universidad desarrollan un sistema llamado DroidChameleon a partir de las herramientas Apktool y Smali/Baksmali para realizar distintas operaciones de evasión de código y ofuscación sobre un malware. En concreto decompilan la aplicación y modifican el binario .dex para compilarla de nuevo con transformaciones hechas en el código, como puede ser el renombre de variables, archivos, reordenar el código, realizar llamadas indirectas, decodificar cadenas de caracteres y matrices, entre otros.

Las conclusiones principales que se obtienen del estudio son:

- Ningún antivirus es por completo eficaz frente a transformaciones simples del código. Por lo que no suponen una solución fiable frente a malware más avanzado.
- En torno al 64% de las firmas que comprueban los antivirus están basadas en la aparición de diferentes elementos en el AndroidManifest.xml de la aplicación. Los objetivos principales que se buscan son: nombres de archivo, nombre de paquete, resúmenes (hash) de la aplicación o información que se puede obtener a través de la API de Android PackageManager.

3.5. Estudio de Técnicas de Monitorización Malware

En cuanto a investigación y desarrollo de herramientas de carácter libre que se dedican a analizar malware, podemos destacar una serie sistemas: (i) basados en servicios web, o (ii) repositorios compartidos por la comunidad. A continuación, se analizan en detalle estos sistemas para determinar qué utilidad pueden a este proyecto.

Actualmente y desde que aparecieron las primeras aplicaciones maliciosas se están desarrollando herramientas que permiten conocer qué hacen estas aplicaciones en el entorno de Android y cómo se comunican con el exterior. Lo más importante que se pretende obtener de estos sistemas es:

- Conocer a través de ejemplos de uso qué elementos de análisis comparten estas herramientas, para focalizar el sistema de replicación en las características que más comúnmente definan un malware.
- Conocer en profundidad el funcionamiento de las herramientas y realizar un estudio que determine si alguna de ellas se pudiese adaptar a la monitorización del sistema de replicación en la cloud.

Después de buscar información sobre distintos métodos de análisis, sistemas de ingeniería inversa y sistemas de detección tanto estáticos como dinámicos, se presentan las principales herramientas con relevancia para el sistema:

AndroGuard	Herramienta de ingeniería inversa para análisis estático
Anubis	Sitio web para el análisis malware de binarios
ApkInspector	Herramienta de ingeniería inversa para análisis malware
ApkTool	Herramienta de ingeniería inversa que permite reempaquetar con modificaciones
DroidBox	Herramienta de análisis dinámico de aplicaciones malware
Mobile Sandbox	Sitio web para el análisis malware de aplicaciones Android
SandDroid	Sitio web para el análisis malware de aplicaciones Android
VirusTotal	Sitio web para el análisis de binarios basado en meta-antivirus

Tan solo se va a entrar en detalle en las herramientas de escritorio y que por tanto se podrían utilizar para crear el sistema.

Las herramientas que se alojan en servidores web tan solo van a servir a modo de orientación para dar una idea de qué se elementos, actividades o eventos se pueden monitorizar en una aplicación para determinar si es malware o no.

3.5.1. AndroGuard

Es la herramienta de escritorio más completa y útil, a pesar de que no realiza análisis dinámico. Androguard toma como entrada una aplicación apk y la desgrana de tal manera que es posible obtener entre otras cosas: los archivos fuente .java, el documento AndroidManifest.xml, la aplicación en formato DVM (Dalvik Virtual Machine), un listado con todos los métodos usados en el código, etc. Este sistema está implementado en el lenguaje Python, y conforma una serie de librerías que otorgan gran funcionalidad para el análisis de ingeniería inversa de aplicaciones. Los principales componentes son:

- ***Androaxml***: Obtiene una simplificación del *AndroidManifest* de cualquier aplicación para que sea fácilmente legible.
- ***Androapkinfo***: Obtiene información básica relevante como archivos contenidos en la aplicación, permisos, actividad principal, servicios, etc.
- ***Androcsign***: Permite crear firmas para la base de datos de malware en base a métodos o clases específicas.
- ***Androdd***: Genera gráficos con la relaciones que existen entre cada método de cada clase.
- ***Androdiff***: Se usa para comprobar el parecido que existe entre dos aplicaciones, por si una es base de la otra.
- ***Androgexf***: Genera un gráfico de la aplicación en formato *gexf*.
- ***Androlyze***: Permite mediante línea de comandos interactiva analizar la aplicación.

A continuación se muestra un ejemplo de uso de Androguard, donde la variable *d* es un objeto del archivo en formato DVM ya decompilado del cual podemos obtener el código fuente de la clase que se requiera.

```
In [3]: d.CLASS_Lsergio_samples_internet_MainActivity.source()
package sergio.samples.internet;
public class MainActivity extends android.app.Activity {
    final private static String DEBUG_TAG;
    private android.widget.Button boton;
    private android.widget.TextView textView;
    private android.widget.EditText urlText;
    static synthetic String access$0(sergio.samples.internet.MainActivity p1, String p2)
    {
        return p1.downloadUrl(p2);
    }
    static synthetic android.widget.TextView access$1(sergio.samples.internet.MainActivity p1)
    {
        return p1.textView;
    }
    private String downloadUrl(String p8)
    {
        v0 = new java.net.URL(p8).openConnection();
        v0.setReadTimeout(10000);
        v0.setConnectTimeout(15000);
        v0.setRequestMethod("GET");
        v0.setDoInput(1);
        v0.connect();
        v0.getResponseCode();
        v2 = v0.getInputStream();
        v1 = this.readIt(v2, 500);
        if (v2 != 0) {
            v2.close();
        }
        return v1;
    }
    public void myClickHandler(android.view.View p7)
    {
        v2 = this.urlText.getText().toString();
        v1 = this.getSystemService("connectivity").getActiveNetworkInfo();
        if ((v1 == 0) || (v1.isConnected() == 0)) {
```

Figura - 14 - Androguard

3.5.2. Anubis

Anubis Iseclab es una asociación de investigadores del campo de la seguridad informática que han creado un portal web al cual se pueden subir tanto aplicaciones Android como ejecutables de Windows para analizarlos y crear informes que muestren si contiene actividad maliciosa. La información que genera Anubis es muy variada ya que realiza tanto análisis dinámico como estático. En el análisis estático, aparte de la información general como nombre de la aplicación, hashes, tamaño del archivo y nivel del API, genera datos como: lista de actividades, *Broadcast* Receivers, permisos requeridos en el AndroidManifest.xml, permisos que utiliza realmente, características usadas como la localización GPS y por último las direcciones URL en claro que contiene el código de la aplicación.

En cuanto al análisis dinámico, se recoge la siguiente información:

- Operaciones de fichero: bien si son lectura, escritura, el tiempo que tarda y la ruta completa donde se ha realizado la operación.
- Operaciones de red: donde se recoge el tiempo en el que se ha hecho la operación, si ésta es de abrir conexión o solicitar información (GET), el host al que se realiza, y el puerto.
- Servicios empezados: como por ejemplo servicio de recibo de SMS.
- Fuga de datos: tiempo en el que se ha realizado, el tipo de fuga como red o SMS, contenido de información de la fuga como número de teléfono, IMEI, etc. Y por último el destino.

Figura - 15 - Anubis

3.5.3. ApkInspector

Es una herramienta similar a AndroGuard pero que permite, gracias a su interfaz, ver de manera más cómoda y clara el interior de una apk. Como por ejemplo el archivo Android-Manifest. De aquí se obtiene información básica como son:

- Paquete

- Recibidores
- Servicios
- Permisos

Permite visualizar las clases que posee la aplicación, por lo que se puede ver un árbol con los métodos existentes. También permite ver las variables de tipo String y su valor específico.

Resulta una herramienta muy potente ya que su funcionalidad está basada en la herramienta Androguard, pero la manera de mostrarlo en una interfaz resulta mucho más cómoda.

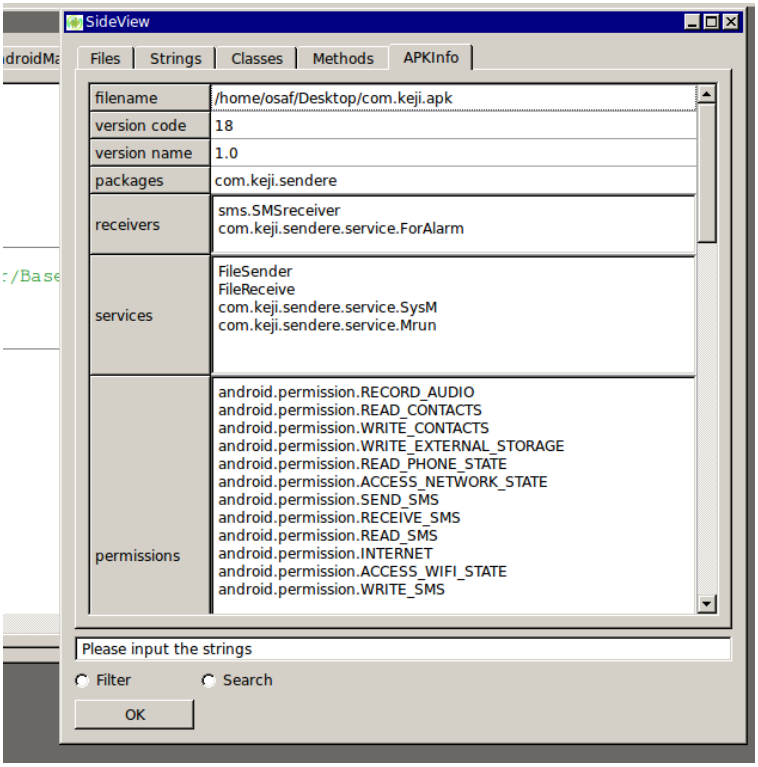


Figura - 16 - ApkInspector

3.5.4. ApkTool

Se trata de una herramienta para depurar una aplicación. Decompila y empaqueta de nuevo la aplicación pero con el código entero modificado (las parte de implementación de los métodos) con Smali y crea una copia de la aplicación apta para depurar la ejecución. Es por lo tanto una herramienta que a priori no genera logs con información relevante para determinar si una aplicación es maliciosa o no. En cambio puede ser una herramienta bastante interesante de estudiar en caso de que se desee analizar una aplicación que sepamos a ciencia cierta que es maliciosa, por lo que se podría depurar de manera segura y estudiar su comportamiento.

```
pimp@pepino ~/HerramientasTFG/apktool1.5.2/Internet/smali/sergio/samples/internet $ cat MainActivity.smali
.class public Lsergio/samples/internet/MainActivity;
.super Landroid/app/Activity;
.source "MainActivity.java"


# annotations
.annotation system Ldalvik/annotation/MemberClasses;
    value = {
        Lsergio/samples/internet/MainActivity$DownloadWebpageTask;
    }
.end annotation


# static fields
.field private static final DEBUG_TAG:Ljava/lang/String; = "HttpExample"


# instance fields
.field private boton:Landroid/widget/Button;

.field private textView:Landroid/widget/TextView;

.field private urlText:Landroid/widget/EditText;


# direct methods
.method public constructor <init>()V
    .locals 0

    .prologue
    .line 28
    invoke-direct {p0}, Landroid/app/Activity;-><init>()V

    return-void
.end method
```

Figura - 17 - Apktool

3.5.5. Droidbox

DroidBox es una herramienta que através de un AVD (Android Virtual Device) es capaz de monitorizar diversas características de una aplicación. Su funcionamiento es algo complejo, ya que sustituye la imagen del kernel que existe en el Android-SDK por una propia que contiene el código necesario para que se generen los logs. Inicialmente se realiza un análisis estático que genera los resúmenes de la aplicación, lectura de permisos en el AndroidManifest además de servicios y *Broadcast* usados. La potencia de DroidBox radica en la lectura e interpretación del LogCat que genera el sistema Android para determinar a un bajo nivel, qué operaciones se está realizando el sistema operativo ordenadas por la aplicación (análisis dinámico). Además genera gráficas e histogramas para mostrar el comportamiento de la app que se desea monitorizar. Para que funcione se debe tener instalada la librería de gráficas de Python:

"*apt-get install python-matplotlib*"

El protocolo de funcionamiento es el siguiente:

- Se levanta el dispositivo virtual con el comando:
  "*./startemu.sh avdName*"
- Se inicia la monitorizacion con el comando:
  "*./droidbox.sh appName.apk segundosDeMonitorizacion*"

Durante la ejecución se debe interactuar con la aplicación para generar comportamiento y en caso de existir malware darle pie a que comience a funcionar, ya que hay cierto malware que hasta que no ocurre un evento específico se mantienen inactivos. Se puede utilizar Monkey para generar esta interacción. La información que esta herramienta analiza es:

- Operaciones de lectura/escritura.
- Actividades criptográficas. Utilización de API Criptográfica de Java.
- Actividades de red. Como apertura de conexiones y trafico saliente.
- Intent Receivers.
- Permisos declarados.
- Fuga de información.
- Envío de SMS.
- Realización de llamadas.

Es la única herramienta de escritorio de las estudiadas que realiza análisis dinámico de aplicaciones, por lo que sería muy interesante integrarla en el sistema de replicación de acciones en la cloud que se va a desarrollar.

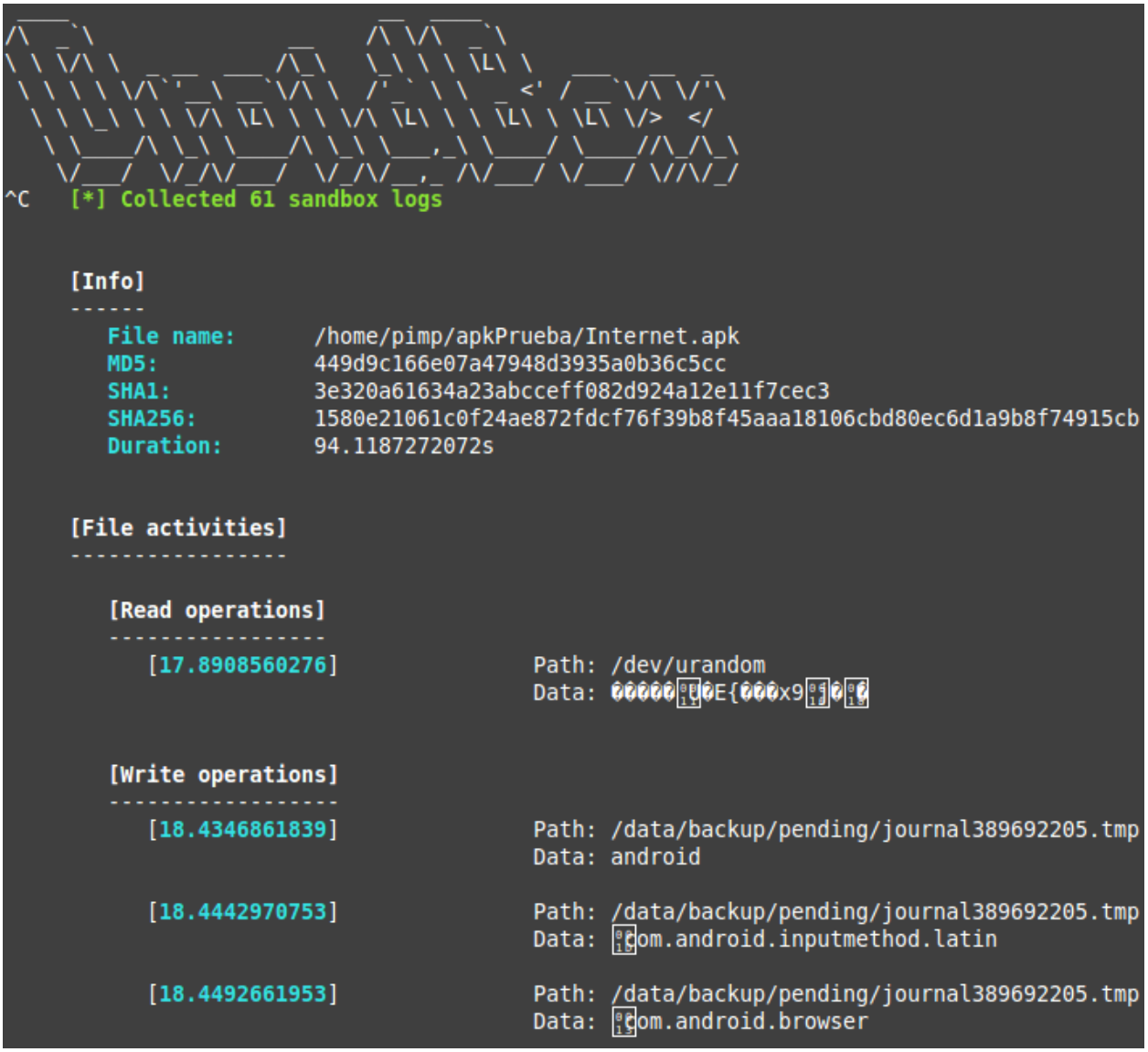


Figura - 18 - DroidBox

3.5.6. Mobile SandBox

Tan solo en el año 2012 en esta web se analizaron más de 300.000 aplicaciones nuevas descargadas de diversos markets de Android (lícito y no lícitos), de las cuales 43.000 aplicaciones eran maliciosas, perteneciendo a 115 familias distintas. La mayoría de estas aplicaciones se descargaron de markets rusos y asiáticos. Pero lo que llama la atención es que se encontraron 13 familias de malware de aplicaciones descargadas del market oficial de Google (Google Play Store).

Esta herramienta es muy similar a Anubis ya que de la misma manera analiza una aplicación Android. Genera logs por separado pero versión HTML, por lo que no se pueden descargar como XML. Genera información general de la aplicación, un análisis estático, uno dinámico, un análisis muy exhaustivo del tráfico de red (PCAP Analysis), y además el resultado de ser

analizada la aplicación por el motor de detección Virus Total. Este último resultado sólo es un porcentaje de antivirus que han detectado la aplicación como malware (12/43, por ejemplo). La información general de la aplicación son los resúmenes sha265 y md5, el nombre de la aplicación, el paquete de la misma y la versión mínima del SDK. En cuanto al análisis estático, se muestra:

- Características usadas.
- Permisos requeridos de Android-Manifest.
- Permisos usados.
- Llamadas a APIS para permisos usados.

Un ejemplo del log generado por Mobile Sandbox es:

Sample SHA256:8aeba630bb19a47102b0d048906a0f0316be315c

1a7405824e4a35ceb8169946

Sample MD5:7503128d14fa8fc6b9b64ce6e9cd90e3

Start of Analysis:April 1, 2012, 11:48 a.m.

End of Analysis:April 1, 2012, 11:48 a.m.

Used Features:

- *android.hardware.location*
- *android.hardware.location.network*
- *android.hardware.wifi*
- *android.hardware.touchscreen*

Requested Permissions from Android Manifest:

- *android.permission.INTERNET*
- *android.permission.ACCESS_NETWORK_STATE*
- *android.permission.ACCESS_WIFI_STATE*
- *android.permission.READ_PHONE_STATE*
- *android.permission.ACCESS_COARSE_LOCATION*

Used Permissions:

- *android.permission.INTERNET*
- *android.permission.ACCESS_NETWORK_STATE*
- *android.permission.ACCESS_WIFI_STATE*
- *com.android.browser.permission.READ_HISTORY_BOOKMARKS*
- *android.permission.READ_PHONE_STATE*
- *android.permission.READ_LOGS*

Responsible API calls for used Permissions:

- *java/net/HttpURLConnection*
- *android/content/Context;->startActivity*
- *android/net/ConnectivityManager;->getActiveNetworkInfo*
- *android/net/wifi/WifiManager;->getConnectionInfo*
- *android/provider/Browser;->getAllVisitedUrls*

- *android/telephony/TelephonyManager;->getDeviceId*
- *java/lang/Runtime;->exec*

Used Intents:

- *android.intent.action.MAIN*
- *android.intent.category.LAUNCHER*

Used Activities:

Potentially dangerous Calls:

- *printStackTrace*
- *HttpURLConnection*
- *HTTP GET/POST*
- *getSystemService*
- *getDeviceId*
- *Execution of native code*
- *HttpPost*
- *getSubscriberId*

Used Services and Receiver:

- *UpdateCheck*
- *Used Providers:*
- *Used Networks:*

Found URLs:

- *http://proxy.youdraw.cn/api/proxy*
- *http://www.facebook.com/otothel.apps*
- *http://www.facebook.com/otothel.apps*
- *http://r2.adwo.com/adweb*
- *http://r.domob.cn/a/*
- *http://e.domob.cn/report*
- *http://ad.gongfu-android.com:7500/ad/nadp.php?v=1.3&id=all*
- *http://cast.ra.imocha.com/p/?pid=*
- *http://amob.acs86.com/a.htm?pv=1&sp=*
- *http://ad.gongfu-android.com:7500/ad/nweb.php?v=1.3&u=*
- *http://ad.gongfu-android.com:7500/ad/nadp.php?v=1.3&id=*

Se puede ver el log completo del malware "com.atools.cuttherope-LeNa.b.apk" buscando el código: 7503128d14fa8fc6b9b64ce6e9cd90e3 en la web.

3.5.7. Sandroid

Es una herramienta alojada en un servidor web que genera un log algo escaso en cuanto a funcionalidad maliciosa de la aplicación, pero que genera unos gráficos muy interesantes con técnicas de clasificación de Aprendizaje Automático, como J48, ID3, NaiveBayes.

No hay que destacar nada interesante o que no se haya visto con otra herramienta. Aunque es capaz de detectar la ejecución de comandos nativos como chmod, al buscar el uso del método getRuntime().

El log generado para la aplicación analizada es el siguiente:

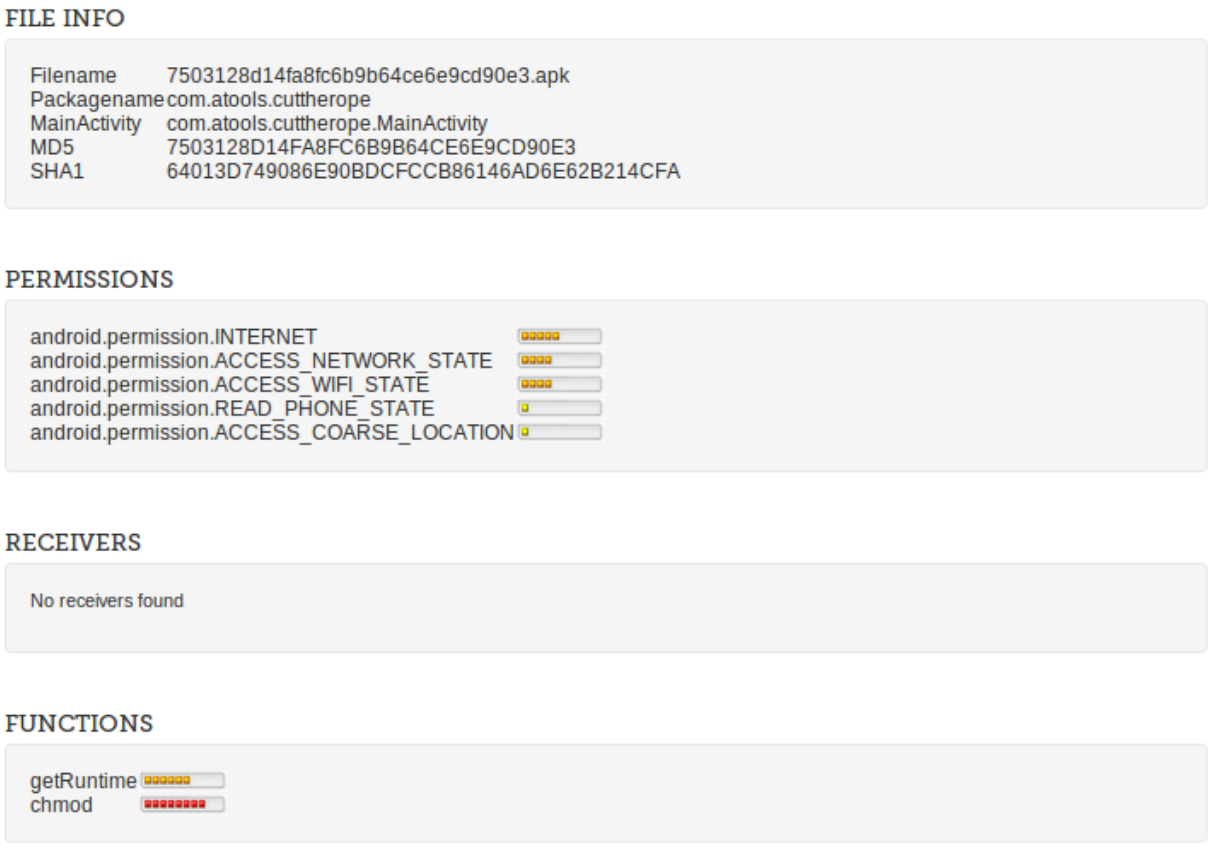


Figura - 19 - Sandroid

3.5.8. VirusTotal

Virus Total es una web que nació en 2004 y fue creada por Hispasec Sistemas. La finalidad de esta web es permitir a los usuarios subir todo tipo de archivos que sean susceptibles de contener código malicioso para realizar un análisis malware. Virus Total utiliza el motor de análisis de gran cantidad de antivirus y herramientas antimalware como:

- AVG
- Avast
- Avira
- Fortinet
- F-Secure
- Kaspersky
- Malwarebytes
- McAfee
- Symantec
- TrendMicro

VirusTotal se puede considerar un meta-antivirus al analizar un archivo con el conjunto de antivirus mostrado. El resultado que es ordenado por cada uno de los sistemas con los que tiene acuerdos y cada uno de ellos muestra si el archivo es considerado o no malware. Si lo es, indica el nombre o la familia a la que pertenece.

Por lo tanto, el valor que se proporciona es realizar un porcentaje de probabilidad de que el archivo sea o no considerado malware en función de los resultados de todos los análisis individuales.

En el periodo de tiempo que VirusTotal lleva funcionado ha reunido un gran número de muestras, por lo que la base de datos es enorme y completa. Esto ha hecho que una gran compañía como Google se interese por ellos, hasta el punto de comprar la empresa en 2012.

Para nuestro proyecto hemos realizado el análisis de una muestra, en concreto los malware:

- *AndroidOS_DROIDKUNGFU.B*
- *Backdoor.AndroidOS.KungFu.z*

A pesar de que sólo proporcione el nombre del malware y no ningún comportamiento u otra información relevante de la aplicación puede ser muy útil para saber rápidamente si se trata de un virus conocido y de qué tipo es.

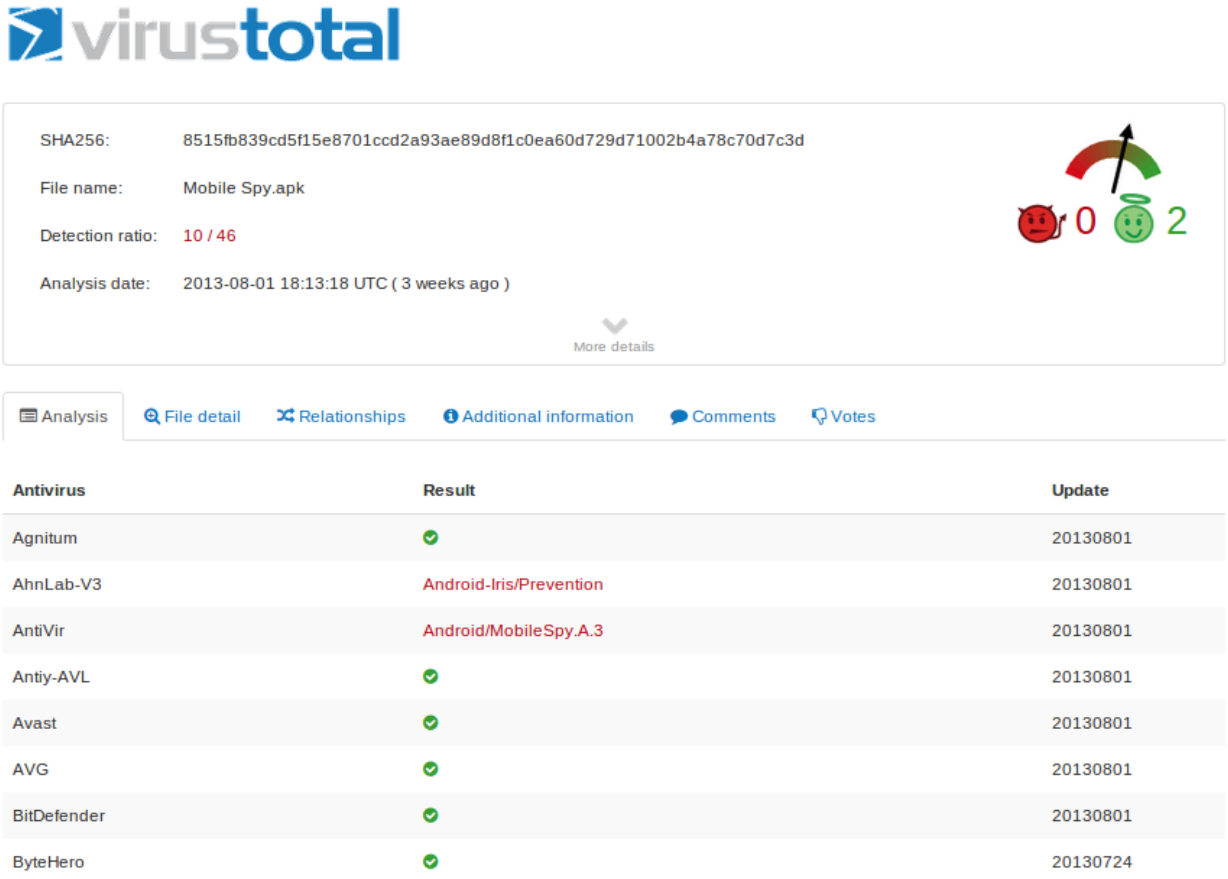


Figura - 20 - VirusTotal

3.6. Estudio de Técnicas de Replicación en la Cloud

A continuación se realiza un análisis de las principales herramientas y técnicas estudiadas relativas a la replicación de actividad de terminales móviles en la nube. Se trata de una serie de papers que investigan cómo se puede abordar el actual problema del estudio del comportamiento de un dispositivo móvil debido a la limitación de recursos que éste tiene, desplazando el estudio a la nube con técnicas ágiles de monitorización y replicación.

3.6.1. CloneCloud

CloneCloud [21] traslada la ejecución de aplicaciones móviles de un dispositivo a un cloud o máquinas virtuales clonadas que de manera eficiente distribuyen las tareas de la ejecución para aliviar la carga de procesado en el dispositivo físico.

El objetivo principal es aumentar el rendimiento de ejecución de aplicaciones y liberar la carga del consumo de recursos en el dispositivo, para posteriormente al procesado, devolver los resultados obtenidos al terminal móvil.

Para realizar la ejecución de manera eficiente se realizan particiones de los procesos de las aplicaciones utilizando técnicas de perfilado de aplicaciones dinámicas y estáticas. Esto permite distribuir la carga de procesado entre varias máquinas de la cloud.

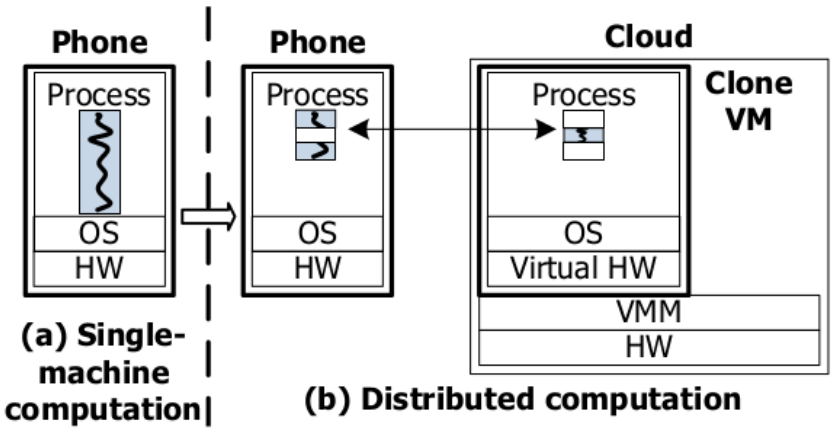


Figura - 21 - CloneCloud Distributed computation

3.6.2. ThinkAir

Esta herramienta [22] crea un marco de trabajo para los desarrolladores de aplicaciones móviles que permite migrar la carga de ejecución de las aplicaciones del terminal a la nube. Está orientado a descargar el peso del procesamiento, y por tanto el ahorro de recursos, de la ejecución de aplicaciones que pueden poseer gran carga computacional.

ThinkAir se integra en la aplicación móvil del desarrollador utilizando su propia API que permite gestionar la ejecución del código y los datos a través de un flujo de ejecución en la nube.

Este sistema está basado en CloneCloud y en MAUI[6], mejorando la funcionalidad de ambos estudios. Permite escalar la carga computacional de las aplicaciones al número idóneo de máquinas virtuales en la nube y gracias a la API permite realizar la gestión y flujo de ejecución desde el terminal hasta la nube, siendo una descarga del procesamiento por demanda del desarrollador.

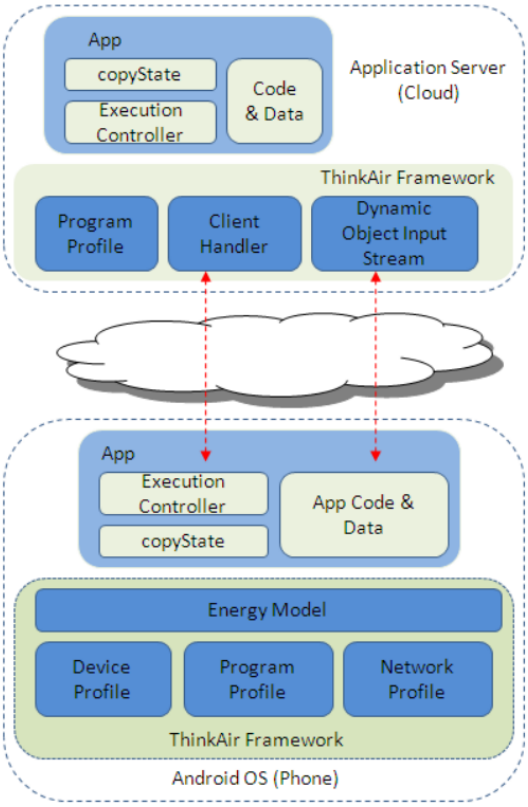


Figura - 22 - ThinkAir Framework

3.6.3. ParanoidAndroid

El sistema ParanoidAndroid [12] está basado en la replicación del comportamiento en tiempo real de un terminal físico a uno virtualizado en un servidor. El objetivo es poder analizar en el dispositivo virtualizado todo el comportamiento replicado desde el terminal físico y poder detectar en tiempo real anomalías o comportamiento malicioso.

En el apartado de motivación se da un breve funcionamiento de este sistema, el cual sentó las bases para el presente trabajo.

[6] MAUI

3.6.4. Secloud

Este sistema conforma un nuevo entendimiento de los antivirus para smartphone. Mientras que las aplicaciones de análisis instaladas en los dispositivos consumen gran cantidad de recursos, como memoria, almacenamiento, CPU o batería para realizar las tareas de análisis, esta solución traslada por completo el peso del análisis al cloud que se aloja en servidores de gran potencia para realizar de manera eficiente un análisis del sistema. Las tareas principales que propone:

- Análisis en búsqueda de virus.
- Monitorización de llamadas al sistema.
- Integridad de archivos.
- Sistema de detección de intrusiones.

Secloud es una aplicación ligera que replica en tiempo real el comportamiento de bajo nivel de un smartphone al cloud de análisis. En caso de detectar comportamiento malicioso el servidor produciría una comunicación con el agente Secloud instalado para restringir o limitar la actividad del malware. En concreto Secloud realiza las siguientes tareas:

1. Recolectar información generada por el usuario y sensores desde las distintas interfaces del dispositivo y las envía a la nube.
2. Gestionar las redes del dispositivo utilizando un proxy de comunicaciones.
3. Permanecer a la espera de notificaciones provenientes de la nube para ejecutar las acciones solicitadas.

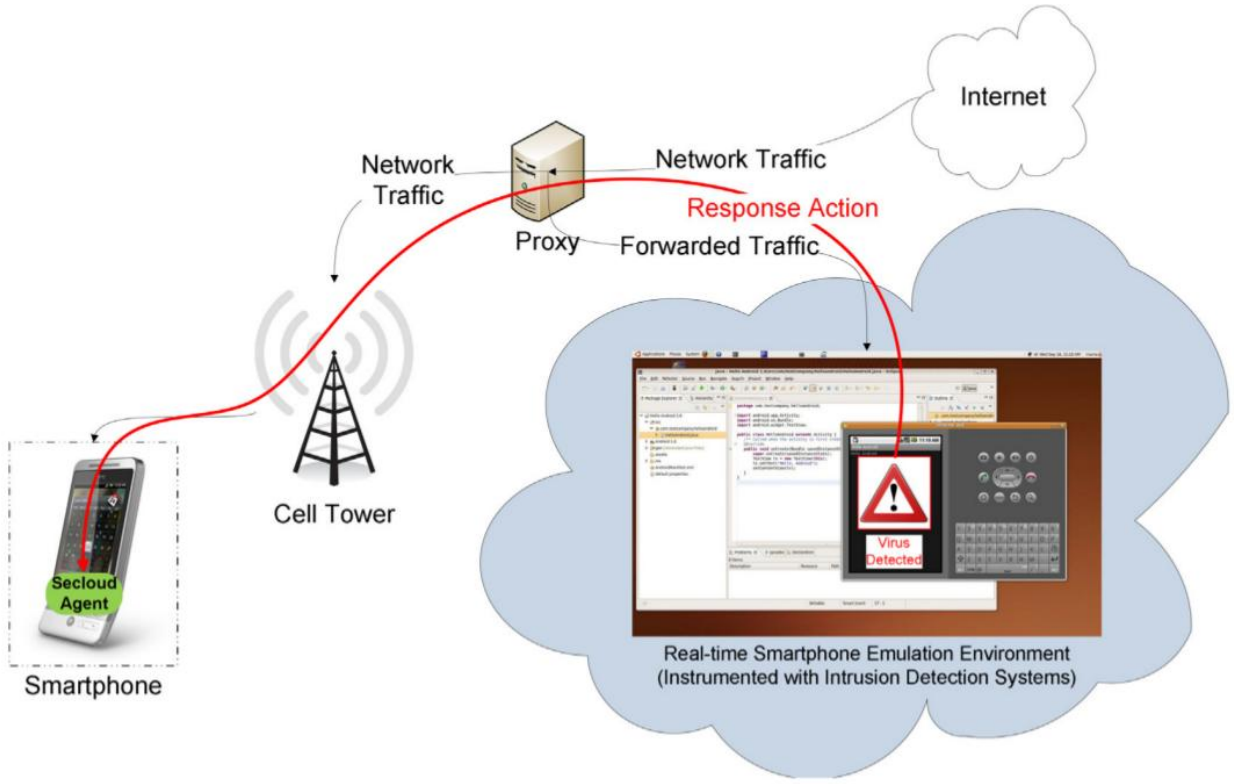


Figura - 23 - Secloud architecture

3.6.5. CosecCloneCloud

CosecCloneCloud es un sistema desarrollado por Guillermo Suarez de Tangil (investigador en el departamento de COmputer SECurity de la Universidad Carlos III) basado en el paper Paranoid Android [12] para la replicación de acciones de un terminal a un dispositivo virtual. Está desarrollado prácticamente en su totalidad en Python y se integra con el Android SDK, ya que utiliza el android SDK Manager para la gestiona de APIs de Android además de las herramientas ADB o emulator.

La arquitectura es la siguiente:

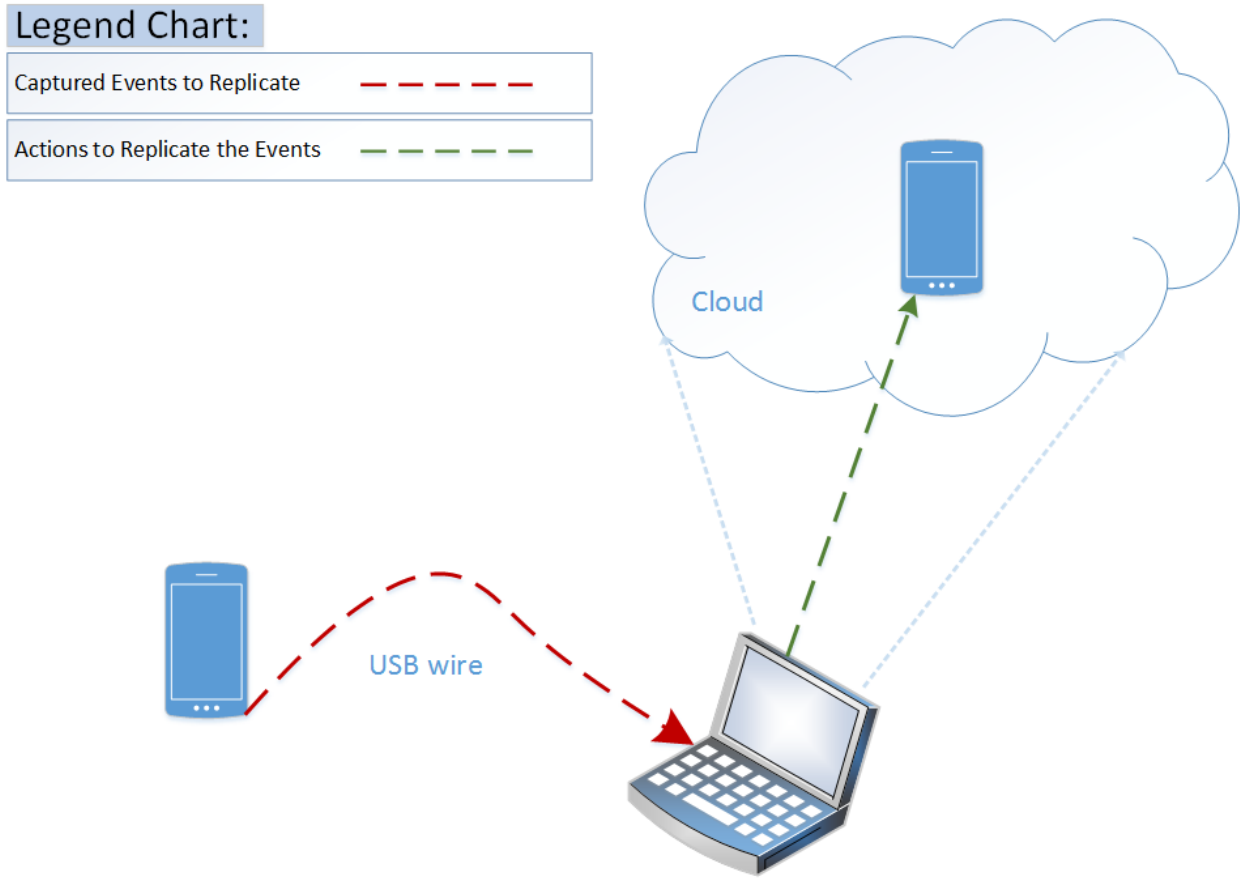


Figura - 24 - CosecCloneCloud

Se observa el terminal Android con versión 2.3 que va a generar eventos según las acciones que realice el usuario. El equipo es un PC o servidor que levanta un dispositivo android virtual (cloud) donde se van a replicar las acciones del terminal físico.

El sistema inicial está limitado por conectividad a un cable USB que conecta el terminal con el servidor. A través de adb, y más concretamente con el comando "*adb -d logcat -c && adb -d logcat*" se capturan los eventos de sistema que son generados en el terminal.

La implementación en Python *parsea* cada evento de entrada al sistema, y en caso de que exista una template de acción para tal evento, se replica la acción según lo implementado dentro de la template. La acción no está definida ni estandarizada, ya que existe gran cantidad de opciones a la hora de generar acciones en un dispositivo Android, y más fácilmente en uno virtual ya que no tiene la limitación de permisos al no ser un dispositivo de producción.

La herramienta tiene una única template que se corresponde con el evento del tipo ActivityManager, por lo que el nombre de la clase es ActivityManager, que implementa como parámetro un objeto RunnerEvent que, una vez establecida la acción, se ejecuta llamando al método "*clone_call()*" que ejecuta a través de la herramienta Monkey de Android la actividad definida dentro de la clase ActivityManager. De esta manera se pueden replicar en la cloud las acciones generadas por el gestor de actividades del sistema Android.

El parseo realizado por la template tiene en cuenta eventos generados únicamente por la versión de Android 2.3. EL sistema no tiene limitación frente al cloud, ya que la acción replicada por la template es estándar a todas las APIs de Android.

En el estado inicial del sistema el resto de eventos que genera el terminal no se contemplan, por lo que se ignoran hasta que se creen plantillas nuevas con acciones de replicación de eventos.

Esta herramienta está pensada para ser mejorada con la incorporación de nuevas plantillas que parseen más eventos de sistema para generar acciones nuevas en la cloud.

El sistema implementado en presente proyecto está basado en CosecCloneCloud.

Sección 4

ESTUDIO DE VIABILIDAD

4.1. Objetivos

Antes de abordar un proyecto de estas dimensiones es necesario realizar un análisis que permita valorar la viabilidad del mismo teniendo como inputs el sistema requerido por el cliente y el alcance según los recursos disponibles. Todo ello teniendo en cuenta el estudio que se ha realizado sobre el estado del arte de Android, el malware existente y las herramientas de monitorización disponibles.

Como parte del estudio de viabilidad, en la gestión del proyecto se deben definir los roles involucrados en el desarrollo del mismo, identificando en caso necesario cuestiones tan importantes para el buen desarrollo del proyecto como la metodología a utilizar, las tareas y responsabilidades de cada rol, identificación de costes, marco de comunicación y métricas de calidad.

Una vez definido y acotado el sistema para que el proyecto sea viable se realizará un Plan de Trabajo que marque unas fechas significativas para el cumplimiento de las tareas identificadas en el proyecto. El calendario de trabajo con las tareas y el responsable de cada una de ellas se elaborará en un Diagrama de Gantt que facilite el seguimiento y la calidad del proyecto.

4.2. Definición del Sistema

En este apartado se definen a un alto nivel cuales son las necesidades que requiere el cliente para su sistema. En base a ello y al estudio del arte se define el alcance real del proyecto.

El objetivo de realizar esta definición del sistema es ceñirse a una solución real y factible que pueda cumplir con los requisitos del cliente. Esta definición se utilizará como base para el análisis en profundidad del sistema.

4.2.1. Necesidades del cliente

A continuación se definen a un alto nivel las necesidades del cliente que se han identificado a través de entrevistas y reuniones. Suponen un primer acercamiento de lo que será el sistema en producción.

1. Capturar eventos generados por acciones de un usuario en un terminal físico.
2. Enviar los eventos capturados a un servidor para ser tratados.
3. Que el envío de los eventos sea independiente de la plataforma.
4. Durante la recepción en el servidor los eventos deben normalizarse para implementar acciones correspondientes al evento.
5. Las acciones deben poder ser replicadas en un dispositivo virtual.
6. Implementar un sistema virtual de Android en la nube.
7. Replicar acciones del usuario en base a los eventos generados de un terminal físico a la nube.
8. Incorporar en la nube sistemas de monitorización de aplicaciones malware.
9. Generar el correspondiente informe de análisis de aplicación malware en la nube.

Estas necesidades dan a entender que el sistema deseado tiene un gran peso de innovación, análisis e implementación. En el siguiente apartado de especifica la acotación del proyecto, y las

herramientas con las que se va a desarrollar para satisfacer de la mejor manera posible las necesidades.

4.2.2. Alcance del proyecto

Para acotar las necesidades y que el proyecto sea factible se debe definir inicialmente un alcance real, que pueda satisfacerse con los recursos disponibles para el proyecto. Si posteriormente se desea realizar una revisión del alcance y proponer nuevas metas es motivo de negociación acordar nuevos requerimientos en base a las necesidades no incluidas en el primer alcance.

Teniendo en cuenta las necesidades del cliente y el estudio realizado en el estado del arte, el sistema a desarrollar debe cumplir como mínimo con el siguiente alcance:

- El sistema de replicación de eventos se efectuará reutilizando y optimizando la herramienta CosecCloneCloud. Las acciones a replicar son las típicas que puede realizar un usuario, además las sensibles a monitorizar/utilizar el malware android. Se extenderá sistema para incorporar la replicación de nuevos eventos.
- El sistema debe funcionar correctamente como mínimo con la versión 2.3 de Android, tanto para el terminal físico como para el dispositivo en la cloud.
- Adicionalmente a la conexión USB, el sistema debe funcionar bajo IP.
- Se integrará el sistema de monitorización con Droidbox.

Queda fuera del alcance la inclusión de técnicas de análisis y detección de malware.

En un futuro se podría integrar con otras herramientas para mejorar la funcionalidad, como realizar análisis dinámicos y generar un informe completo estático/dinámico de la aplicación.

4.2.1. Herramientas a utilizar

El proyecto será desarrollado utilizando como base CosecCloneCloud de Guillermo Suarez de Tangil. Actualmente permite la monitorización de un tipo de eventos en un terminal físico, pudiendo replicar la acción al dispositivo virtual.

Esta herramienta es la base del sistema, y al estar implementada en código Python además de con scripts de configuración en bash shell de Linux, existe una restricción inicial de infraestructura donde se debe desarrollar todo el sistema en un entorno Linux, junto con la continuación y mejora de CosecCloneCloud en python.

Adicionalmente, se integrará con Droidbox para la monitorización de aplicaciones malware, cuya implementación también está basada en python.

Para que el sistema funcione bajo IP, debe existir una conexión WiFi por la cual se puedan replicar los eventos al servidor. En un primer acercamiento al análisis de esta funcionalidad, se intentará realizar una estructura de cliente- servidor, donde se genere bajo lenguaje C estas aplicaciones. La primera, el cliente que envía información, debe instalarse en el terminal Android para escuchar los eventos del terminal y enviarlos al servidor. La segunda, la herramienta del servidor debe permitir la comunicación con el cliente para escuchar los eventos que envía y ponerlos como inputs en el sistema CosecCloneCloud.

Como conclusión del estudio previo de la herramienta CosecCloneCloud, se puede identificar una limitación en cuanto a cantidad de eventos capturados, que viene dada por el propio logcat que tan solo proporciona eventos del sistema. Si es necesario para capturar acciones del usuario en el terminal se estudiará la posibilidad de desarrollar aplicaciones Android que amplíen la funcionalidad del sistema pudiendo identificar y replicar un mayor número de acciones de usuario.

4.3. Gestión del Proyecto

Para que un proyecto sea factible, exista un orden y no se exceda en la utilización de recursos definidos debe existir una gestión mínima que defina cómo se va a desarrollar, la gestión interna, el control de los hitos y el cumplimiento de las fechas marcadas.

Este apartado define además las directrices a seguir durante todo el tiempo que dure el desarrollo del proyecto, desde que se comienza a analizar el sistema hasta que se finaliza con la implantación y el mantenimiento.

4.3.1. Metodología de desarrollo

Como se ha comentado en la definición del sistema, partimos de un marco definido como es el de utilización de herramientas existentes e integración y mejora de las mismas. Esto supone un esfuerzo inicial en el aprendizaje de las mismas y en su adecuación a los requisitos del sistema.

Por ello se ha establecido como metodología del desarrollo más adecuada la conocida como modelo incremental, donde el avance del sistema se realiza en pasos cortos, que constan de análisis, diseño e implementación de requerimientos específicos del sistema, para testear el correcto funcionamiento de los mismos.

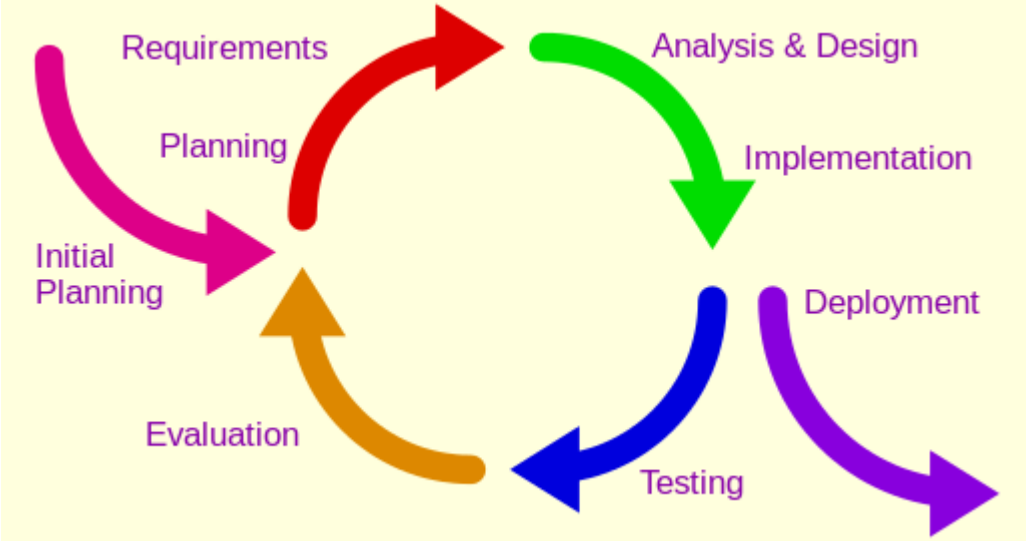


Figura - 25 - Ciclo de vida Iterativo

Esta metodología es necesaria, ya que en un primer momento no se puede asegurar que la factibilidad operacional de todos los requisitos del sistema esté garantizada, por ejemplo, en un análisis inicial no se puede determinar si X acción de usuario se puede capturar en el terminal, o si esa misma acción se puede replicar con las tecnologías con las que se cuenta. Para cumplir con las necesidades del cliente el sistema debe estar basado en la prueba y error de la definición de la arquitectura, el diseño y la implementación.

El desarrollador conoce y asume el riesgo que esto conlleva, ya que la estructura del sistema puede cambiar con suma facilidad.

Tanto el cliente como el equipo de desarrollo son conscientes de que a medida que avance el proyecto la estructura del sistema puede cambiar con suma facilidad. A pesar de ello se asumen los riesgos que puede acarrear esta metodología. El equipo de desarrollo asume por tanto los costes adicionales que puedan surgir del uso de esta metodología liberando al cliente de los mismos.

4.3.2. Stakeholders

Como parte de la gestión del proyecto se debe identificar qué personal estará involucrado en el desarrollo del mismo. Por parte del cliente va a haber un único responsable que se encargará de supervisar el correcto desarrollo del sistema así como de facilitar la información necesaria que el equipo contratado requiera para el desarrollo del proyecto.

El equipo de desarrollo contratado está formado por un único consultor freelancer que será responsable de la realización de todas las tareas de desarrollo del proyecto.

Por tanto la gestión del proyecto estará formada por el siguiente equipo de trabajo:

- **Cliente.** Guillermo Suarez de Tangil.
- **Consultor freelancer.** Sergio Gutiérrez Hernández.

La gestión del proyecto es atómica, estará basada en la comunicación de los dos únicos integrantes del equipo.

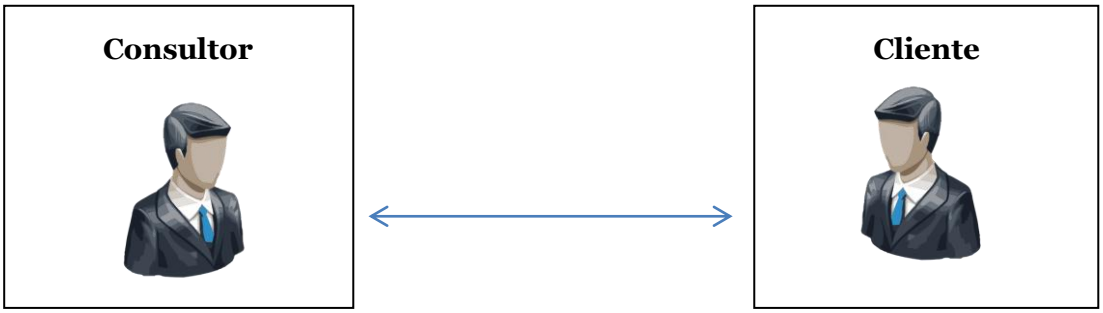


Figura - 26 - Equipo de Proyecto

Las tareas de cada rol son:

Consultor Experto	Cliente
Análisis de la necesidad	**Definición de la necesidad**
Estudio de viabilidad	**Seguimiento del proyecto**
Análisis del sistema	**Resolución de cuestiones**
Diseño del sistema	**Aprobación de tareas**
Desarrollo del sistema	
Evaluación y Pruebas	
Implantación	

Tabla - 2 Definición de Tareas por Rol

4.3.3. Estimación de Costes

Se estima que el proyecto puede realizarse en 4 meses y medio de trabajo de un consultor a jornada completa.

- **Costes asociados a Capital Humano**

Duración		Costes de Capital Humano	
Meses	4	Nombre	Sergio Gutiérrez
Días por mes	20	Cargo	Consultor Experto
Horas por día	8	Salario / Hora	30
Horas totales	*640*	*Coste Total*	*19200*

Tabla - 3 Costes de Capital Humano

- **Costes asociados a elementos Hardware**

PC	
Procesador Intel 2500 k	169
Memoria 4Gb Kingston	54
SSD Crucial M4 128 Gb	130
Monitor 23'' Samsung	140
Otros	200
Coste Total	*693*

Tabla - 4 - Costes HW

- **Costes asociados a Software**

Software	
LinuxMint 15 OS	0
CosecCloneCloud	0
DroidBox	0
Python	0
Eclipse	0
SublimeText2	0
Android SDK	0
MS. Word	50
MS. Project	50
MS. Visio	50
Gimp	0
Coste Total	*150*

Tabla - 5 - Costes SW

- **Costes Totales**

El cálculo de los costes totales viene dado por la suma de los costes Hardware, Software, Capital Humano y otros costes asociados.

El consultor considera hacer una rebaja del coste de los recursos utilizados frente a la amortización realizada con el desarrollo de otros proyectos previos.

Por otro lado, el consultor no ha considerado incluir en el coste de hardware el terminal móvil para la realización de pruebas del sistema, ya que ha utilizado el suyo personal.

Los costes asociados incluyen el alquiler del local de trabajo, abono transporte y material de oficina a utilizar.

	Coste Integro	Reducción por Amortización	Coste Efectivo
Capital Humano	19200	0	19200
HW	693	0.2	554.4
SW	150	0.2	120
Otros	450	0	450
	Coste total del Proyecto (sin IVA)		20493
	I.V.A		21 %
	Coste total del Proyecto (con IVA)		24796.53

Tabla - 6 - CostesTotales del Proyecto

Por lo tanto, los honorarios finales del consultor ascienden a "veinticuatro mil setecientos noventa y seis euros con cincuenta y tres céntimos" (**24 796.53** €).

4.3.4. Diagrama de Gantt

El desarrollo del proyecto está sujeto a un tiempo ideal de 4 meses de trabajo a tiempo completo para una única persona que se encarga de realizar todas las fases del proyecto, desde el análisis hasta la implantación.

Bien si por retrasos en la planificación u otras causas de fuerza mayor no se puede realizar la totalidad del proyecto en el plazo marcado será motivo de negociación realizar una replanificación del mismo, pudiendo acordar con el cliente modificaciones temporales y de requisitos.

A continuación se definen las etapas identificadas de desarrollo del proyecto, y posteriormente se visualiza con el diagrama de Gantt generado.

- **Etapas del Proyecto**

Las etapas del proyecto, con las subtareas identificadas son:

1. Estado del arte
    a. Android
    b. Malware en Android. Evolución y tipos
    c. Antimalware en dispositivos Android
    d. Herramientas de investigación malware
    e. Herramientas Cloud aplicaciones móviles
2. Viabilidad
    a. Definición inicial del sistema.
    b. Gestión del proyecto. Metodología, Equipo de Proyecto y Costes.
    c. Planificación
3. Análisis
    a. Identificación de Requisitos
    b. Eventos generados por Malware
    c. Eventos generados por usuario
    d. Identificación de procesos de sistema
    e. Entorno operativo y herramientas a utilizar
4. Diseño
    a. Sistema de Gestión de Eventos
    b. CosecCloneCloud
    c. Sistema de Replicación en la nube
    d. Alternativas Cloud
5. Implementación
    a. Diseño de la estructura de comunicación
    b. Sistema CloneCloud
    c. Sistema Android
6. Evaluación y Pruebas
    a. Pruebas de requisitos
    b. Pruebas de funcionalidad de replicación de eventos
7. Implantación y mantenimiento
    a. Entorno de producción y dependencias
    b. Publicación del repositorio

Las etapas de Análisis, Diseño e Implementación son iterativas, para satisfacer todos los requerimientos se comprobará en estas etapas la factibilidad del mismo, ya que pueden aparecer restricciones a la hora de diseñar o implementar el sistema.

- **Diagrama de Gantt**

A continuación se añade el diagrama de Gantt con vista de las tareas principales, junto con la relación entre fases para visualizar la iteración entre las mismas. Se puede encontrar el diagrama de Gantt entero, en formato Project en la sección Anexos.

ID	Task Name
1	
2	**Sistema para la monitorización y estudio de aplicaciones maliciosas para Android**
3	**Estado del Arte**
4	Android
5	Malware en Android
6	Antimalware para Android
7	Estudio de herramientas investigación malware
8	Herramientas Cloud de aplicaciones móviles
9	**Viabilidad**
13	**Analisis**
14	Identificacion de requisitos
15	Eventos generador por malware
16	Eventos generados por usuario
17	Identificacion de procesos de sistema
18	Entorno operativo y herramientas a utilizar
19	**Diseño**
24	**Implementacion**
28	**Evaluacion**
31	**Implantacion y Manenimiento**

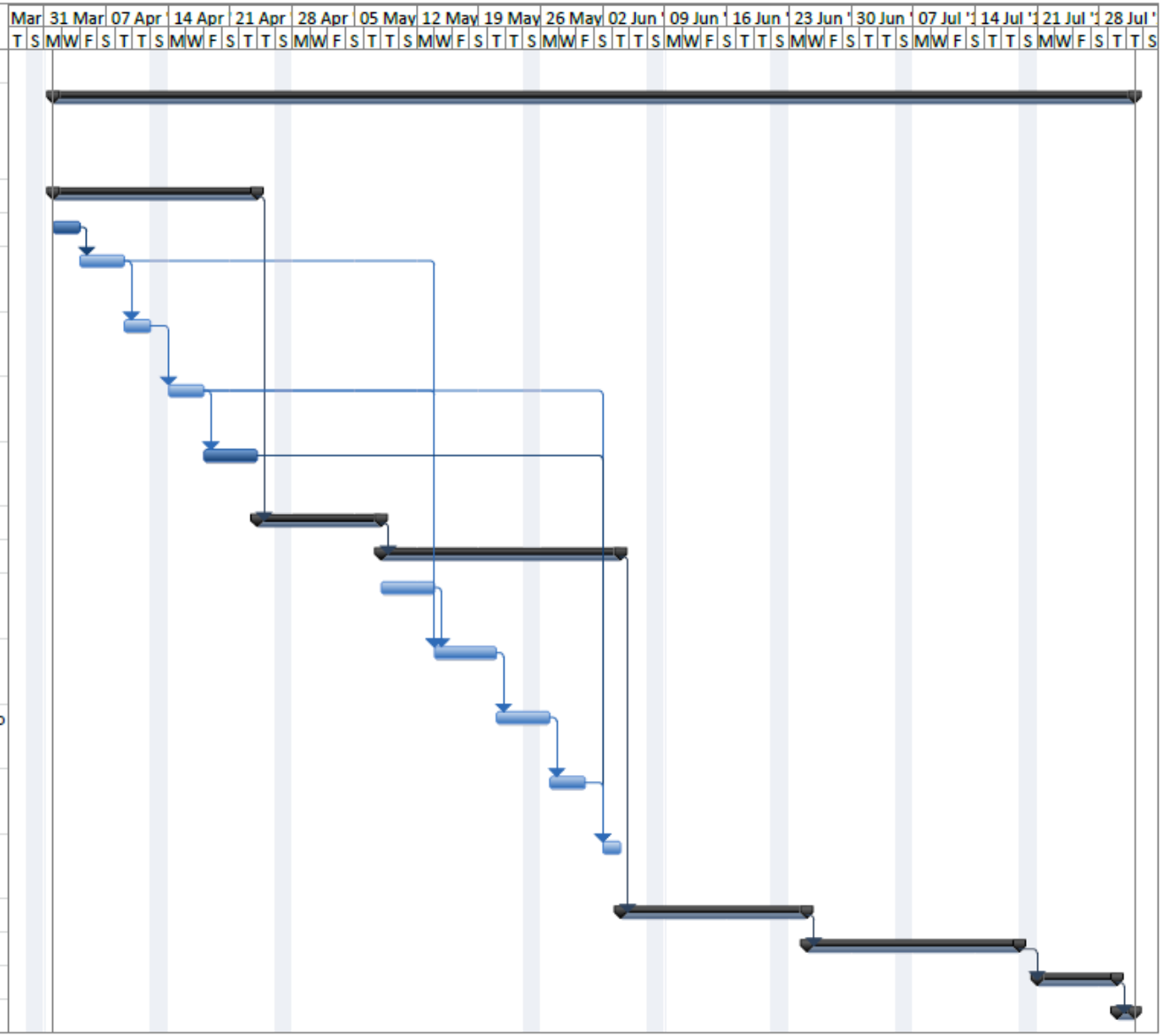


Figura - 27 - Diagrama de Gantt_1

ID	Task Name
1	
2	**Sistema para la monitorización y estudio de aplicaciones maliciosas para Android**
3	**Estado del Arte**
4	Android
5	Malware en Android
6	Antimalware para Android
7	Estudio de herramientas investigación malware
8	Herramientas Cloud de aplicaciones móviles
9	**Viabilidad**
13	**Analisis**
19	**Diseño**
20	Sistema de Gestión de Eventos
21	CosesCloneCloud
22	Sistema de Replicación en la nube
23	Alternativas Cloud
24	**Implementacion**
28	**Evaluacion**
31	**Implantacion y Manenimiento**

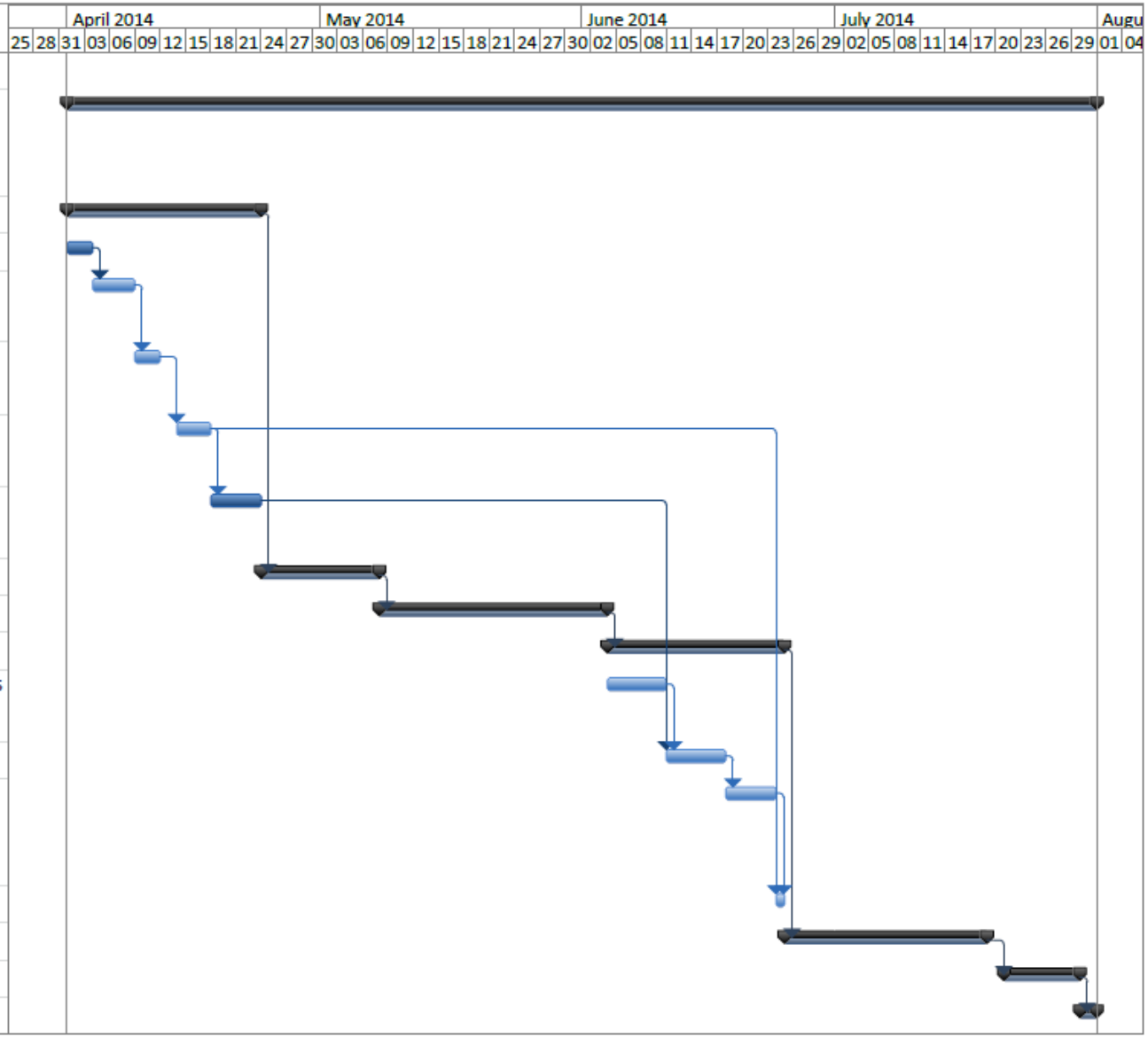


Figura - 28 - Diagrama de Gantt_2

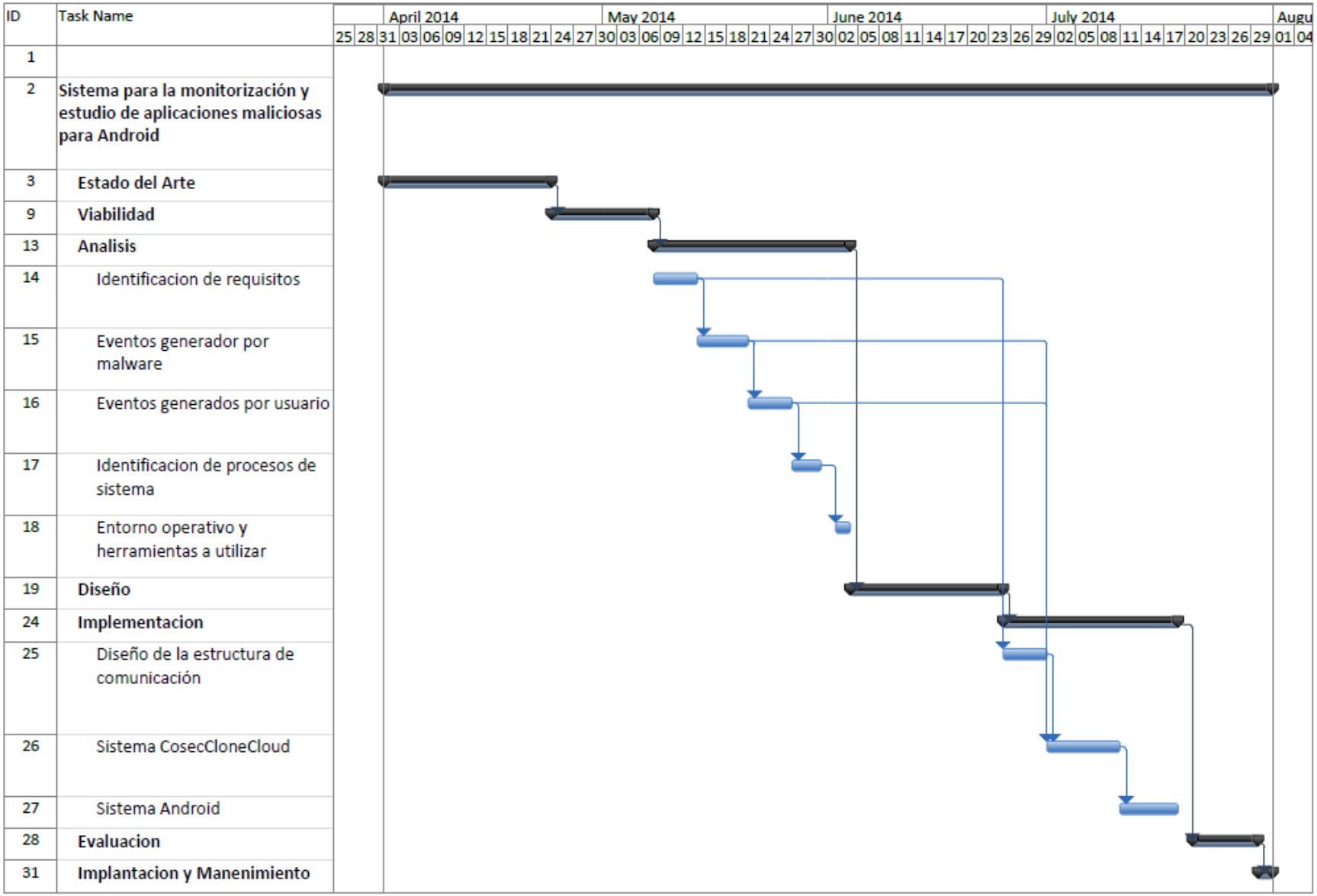


Figura - 29 - Diagrama de Gantt_3

Sección 5

ANÁLISIS

5.1. Introducción al Análisis

Esta etapa del proyecto consiste en un análisis evolutivo del sistema, que se centra principalmente en el asentamiento de los conceptos adquiridos en el estudio del estado del arte, y que se van a adaptar al sistema que se pretende desarrollar.

Por ello, se dedica un apartado a realizar una taxonomía de acciones o eventos que aparecen frecuentemente en aplicaciones malware. Por otro lado se estudiarán las principales acciones que un usuario puede hacer mientras interactúa con un terminal Android, para determinar qué eventos son generados y de qué manera se pueden capturar para que sean inputs del sistema. Después de ello se estudiarán también que herramientas existen en el entorno Unix – Android que permita la replicación de acciones a la cloud.

El catálogo de requisitos será una realimentación de los requisitos de usuario que se definen de manera inicial. Al desarrollar el sistema de manera incremental, surgirán nuevos requisitos una vez estudiado el diseño e intentado la implementación de módulos del sistema. Los requisitos de usuario definen a un alto nivel qué funcionalidad espera el cliente del sistema, así como las limitaciones que ha definido del mismo. Los requisitos de software son una evolución de los de sistema, por lo que cada uno de usuario se desglosará en al menos uno de software.

5.2. Definición de Requisitos

En la fase de análisis deben estar definidos una serie de requisitos que de manera clara y no ambigua expliquen qué debe cumplir e incluso que no debe cumplir el sistema. En caso de existir algún requisito que de manera más técnica determine como debe funcionar, se añadirá al apartado de requisitos no funcionales.

En este proyecto no se van a definir requisitos de negocio o de interfaz de usuario, únicamente de funcionalidad del sistema.

La tabla usada para definir los requisitos es la siguiente:

ID: RU-XX	**TIPO: Tipo Requisito**	**NECESIDAD: Requerido/Deseable/Opcional**	
Nombre	Nombre breve del Req.	Fuente	De quién o qué surge el Req.
Iteración	En qué etapa del ciclo	Verificabilidad	Capacidad de ser testeado
Descripción	Breve descripción del requisito		

Tabla - 7 - Requisitos de Usuario

5.2.1. Requisitos de Usuario

Se detallan a continuación los requisitos de usuario que definen el sistema.

5.2.1.1. Funcionales

ID: RU-01	**TIPO: Capacidad**	**NECESIDAD: Requerido**	
Nombre	Detectar comportamiento	Fuente	Cliente
Iteración	0	Verificabilidad	Alta
Descripción	El sistema debe poder identificar y capturar en tiempo real el comportamiento del terminal físico		

ID: RU-02	**TIPO: Capacidad**	**NECESIDAD: Requerido**	
Nombre	Clasificación del comportamiento	Fuente	Cliente
Iteración	0	Verificabilidad	Alta
Descripción	El sistema debe poder clasificar el comportamiento del terminal físico		

ID: RU-03	**TIPO: Capacidad**	**NECESIDAD: Deseable**	
Nombre	Priorizar comportamiento de malware	Fuente	Cliente
Iteración	0	Verificabilidad	Alta
Descripción	El sistema debe poder identificar y capturar el comportamiento típico del malware en el terminal físico		

ID: RU-04	**TIPO: Capacidad**	**NECESIDAD: Requerido**	
Nombre	Replicar comportamiento	Fuente	Cliente
Iteración	0	Verificabilidad	Alta
Descripción	El sistema debe poder replicar en la cloud el comportamiento		

5.2.1.2. No Funcionales

ID: RU-05	**TIPO: Interfaz HW**	**NECESIDAD: Requerido**	
Nombre	Detectar comportamiento de terminal físico	Fuente	Cliente
Iteración	0	Verificabilidad	Alta
Descripción	El sistema debe identificar y capturar el comportamiento de un terminal físico generado por la acción de un usuario		

ID: RU-06	**TIPO: Interfaz SW**	**NECESIDAD: Requerido**	
Nombre	Programar en lenguaje Python	Fuente	Cliente
Iteración	0	Verificabilidad	Alta
Descripción	El sistema debe estar programado en Python reutilizando el sistema CosecCloneCloud		

ID: RU-07	**TIPO: Interfaz SW**	**NECESIDAD: Requerido**	
Nombre	Capturar eventos y replicar acciones con Android	Fuente	Cliente
Iteración	1	Verificabilidad	Alta
Descripción	El sistema debe capturar eventos generados a través de un middleware en el terminal y replicar las acciones oportunas con una aplicación instalada en la cloud		

ID: RU-08	**TIPO: Rendimiento**	**NECESIDAD: Requerido**	
Nombre	Sistema de detección y replicación flexible	Fuente	Cliente
Iteración	0	Verificabilidad	Alta
Descripción	El sistema debe ser flexible y fácilmente adaptable a la detección de nuevo comportamiento y replicación de nuevas acciones		

ID: RU-9	**TIPO: Interfaz SW**	**NECESIDAD: Deseable**	
Nombre	Sistema independiente de API del terminal	Fuente	Cliente
Iteración	0	Verificabilidad	Alta
Descripción	El sistema debe adaptarse a cualquier versión del sistema operativo del terminal físico		

ID: RU-10	**TIPO: Medio físico**	**NECESIDAD: Requerido**	
Nombre	Sistema independiente de conexión	Fuente	Cliente
Iteración	0	Verificabilidad	Alta
Descripción	El sistema debe funcionar independientemente de la conexión física entre el terminal y el servidor		

ID: RU-11	**TIPO: Rendimiento**	**NECESIDAD: Requerido**	
Nombre	Sistema robusto	Fuente	Consultor
Iteración	1	Verificabilidad	Alta
Descripción	El sistema debe garantizar el correcto funcionamiento en caso de surgir situaciones inesperadas en algún proceso del sistema		

ID: RU-12	TIPO: Rendimiento	NECESIDAD: Deseable	
Nombre	Sistema fluido	Fuente	Consultor
Iteración	1	Verificabilidad	Alta
Descripción	El sistema debe garantizar una fluidez aceptable en los procesos del sistema		

ID: RU-13	TIPO: Rendimiento	NECESIDAD: Deseable	
Nombre	Sistema integrable con otras funcionalidades	Fuente	Cliente
Iteración	0	Verificabilidad	Medio
Descripción	El sistema debe facilitar la integración de otras funcionalidades o mejoras adicionales		

5.2.2. Requisitos de Software

Los requisitos de software son una evolución de los de usuario. Una vez analizado el sistema en profundidad cada requisito de usuario debe ser definido en mayor detalle, y si es necesario debe ser detallado en varios requisitos de software. Por ello la tabla de cada requisito de software tiene identificado el requisito de usuario del que se genera, así se asegura la trazabilidad de los requisitos. La tabla queda de la siguiente manera:

ID: RS-XX	TIPO: Tipo Requisito	NECESIDAD: Requerido/Deseable/Opcional	
Nombre	Nombre breve del Req.	ID Req. Usuario	Req. Usuario del que se genera
		Fuente	De quién o qué surge el Req.
Iteración	En qué etapa del ciclo	Verificabilidad	Capacidad de ser testeado
Descripción	Breve descripción del requisito		

Tabla - 8 - Requisitos de Software

5.2.2.1. Funcionales

ID: RS-01	TIPO: Sistema	NECESIDAD: Requerido	
Nombre	Capturar eventos de sistema	ID Req. Usuario	RU-01
		Fuente	Cliente
Iteración	0	Verificabilidad	Alta
Descripción	El comportamiento del terminal físico se debe capturar a través de los eventos del sistema		

ID: RS-02	TIPO: Sistema	NECESIDAD: Requerido	
Nombre	Generar eventos nuevos	ID Req. Usuario	RU-01
		Fuente	Consultor
Iteración	1	Verificabilidad	Alta
Descripción	El comportamiento que no es enviado como evento de sistema al registro de logs se debe detectar a través de eventos propios		

ID: RS-03	TIPO: Sistema	NECESIDAD: Requerido	
Nombre	Capturados todos los eventos generados	ID Req. Usuario	RU-02
		Fuente	Cliente
Iteración	0	Verificabilidad	Alta
Descripción	Los eventos que genere el terminal, sean de sistema o propios son capturados y actúan de input en el sistema		

ID: RS-04	**TIPO: Sistema**	**NECESIDAD: Requerido**	
Nombre	Tipificar eventos capturados	ID Req. Usuario	RU-02
		Fuente	Cliente
Iteración	1	Verificabilidad	Alta
Descripción	Los eventos capturados deben ser tipificados y en caso de detectar varias acciones disponibles por evento se deben implementar de manera separada		

ID: RS-05	**TIPO: Sistema**	**NECESIDAD: Requerido**	
Nombre	Capturar eventos malware	ID Req. Usuario	RU-03
		Fuente	Cliente
Iteración	0	Verificabilidad	Alta
Descripción	Partiendo del análisis realizado en el estado del arte, se deben capturar y analizar los eventos susceptibles de ser malware (el mayor número posible del Anexo 1)		

ID: RS-06	**TIPO: Sistema**	**NECESIDAD: Requerido**	
Nombre	Replicar acciones de usuario en la cloud	ID Req. Usuario	RU-04
		Fuente	Cliente
Iteración	0	Verificabilidad	Alta
Descripción	El sistema debe detectar y replicar las acciones típicas que puede realizar un usuario en el terminal físico		

ID: RS-07	TIPO: Sistema	NECESIDAD: Requerido	
Nombre	Replicar acciones de malware en la cloud	ID Req. Usuario	RU-04
		Fuente	Cliente
Iteración	0	Verificabilidad	Alta
Descripción	Los eventos susceptibles de ser malware deben ser replicados como acción en la cloud		

ID: RS-08	TIPO: Sistema	NECESIDAD: Requerido	
Nombre	Replicar acciones de manera externa	ID Req. Usuario	RU-04
		Fuente	Cliente
Iteración	0	Verificabilidad	Alta
Descripción	La acción o acciones que se puedan replicar de manera externa a la cloud se hará utilizando herramientas que permitan tal hecho		

5.2.2.2. No Funcionales

ID: RS-09	TIPO: Sistema	NECESIDAD: Requerido	
Nombre	Capturar eventos de sistema usando logcat	ID Req. Usuario	RU-01
		Fuente	Cliente
Iteración	0	Verificabilidad	Alta
Descripción	El comportamiento del terminal físico se captura a través de la utilidad logcat		

ID: RS-10	**TIPO: Sistema**	**NECESIDAD: Requerido**	
Nombre	Implementación de acciones desde APK en cloud	ID Req. Usuario	RU-04
		Fuente	Consultor
Iteración	1	Verificabilidad	Alta
Descripción	Las acciones que se deban replicar desde la aplicación Android instalada en la cloud deberán identificarse con un código y enviarse a través de un *Broadcast*		

ID: RS-11	**TIPO: Sistema**	**NECESIDAD: Requerido**	
Nombre	Capturar eventos con ADB	ID Req. Usuario	RU-05
		Fuente	Cliente
Iteración	0	Verificabilidad	Alta
Descripción	Un modo de capturar eventos del terminal se realizará estableciendo conexión con la herramienta Android Device Bridge (ADB)		

ID: RS-12	**TIPO: Sistema**	**NECESIDAD: Requerido**	
Nombre	Lenguaje Python	ID Req. Usuario	RU-06
		Fuente	Cliente
Iteración	0	Verificabilidad	Alta
Descripción	La programación del sistema de replicación se realizará en Python		

ID: RS-13	**TIPO: Sistema**	**NECESIDAD: Requerido**	
Nombre	Integración en bash shell script y Python	ID Req. Usuario	RU-06
		Fuente	Cliente
Iteración	0	Verificabilidad	Alta
Descripción	La integración de los procesos del sistema se realizará en bash shell script o bien Python		

ID: RS-14	**TIPO: Sistema**	**NECESIDAD: Requerido**	
Nombre	Generación de eventos propios	ID Req. Usuario	RU-07
		Fuente	Consultor
Iteración	1	Verificabilidad	Alta
Descripción	El comportamiento del sistema que se requiera capturar como evento, pero que no sea generado automáticamente por el sistema, se deberá capturar por una aplicación instalada en el terminal físico y enviado al log del sistema como evento propio para que se pueda capturar		

ID: RS-15	**TIPO: Sistema**	**NECESIDAD: Requerido**	
Nombre	Replicación de acciones internas	ID Req. Usuario	RU-07
		Fuente	Consultor
Iteración	1	Verificabilidad	Alta
Descripción	Las acciones que no se puedan replicar con herramientas externas se realizarán a través de una aplicación Android instalada en la cloud		

ID: RS-16	**TIPO: Sistema**	**NECESIDAD: Requerido**	
Nombre	Sistema de captura flexible	ID Req. Usuario	RU-08
		Fuente	Cliente
Iteración	0	Verificabilidad	Alta
Descripción	Los sistemas que capturen eventos deben ser fácilmente ampliables sin que suponga un gran coste de modificación		

ID: RS-17	**TIPO: Sistema**	**NECESIDAD: Requerido**	
Nombre	Sistema de replicación flexible	ID Req. Usuario	RU-08
		Fuente	Cliente
Iteración	0	Verificabilidad	Alta
Descripción	Los sistemas que repliquen acciones deben ser fácilmente ampliables sin que suponga un gran coste de modificación		

ID: RS-18	**TIPO: Sistema**	**NECESIDAD: Deseable**	
Nombre	Sistema independiente de API del terminal	ID Req. Usuario	RU-09
		Fuente	Cliente
Iteración	0	Verificabilidad	Alta
Descripción	Los eventos de sistema capturados de distintas versiones de Android se deben homogeneizar en una misma acción		

ID: RS-19	**TIPO: Sistema**	**NECESIDAD: Requerido**	
Nombre	Captura por USB	ID Req. Usuario	RU-10
		Fuente	Cliente
Iteración	0	Verificabilidad	Alta
Descripción	El sistema deberá capturar eventos a través de USB		

ID: RS-20	**TIPO: Sistema**	**NECESIDAD: Requerido**	
Nombre	Captura por WiFi	ID Req. Usuario	RU-10
		Fuente	Cliente
Iteración	0	Verificabilidad	Alta
Descripción	El sistema deberá capturar eventos a través de una conexión inalámbrica		

ID: RS-21	**TIPO: Sistema**	**NECESIDAD: Requerido**	
Nombre	Validación de datos de entrada	ID Req. Usuario	RU-11
		Fuente	Consultor
Iteración	1	Verificabilidad	Alta
Descripción	El sistema validará datos de entrada de los eventos capturados		

ID: RS-22	TIPO: Sistema	NECESIDAD: Requerido	
Nombre	Tratamiento de excepciones	ID Req. Usuario	RU-11
		Fuente	Consultor
Iteración	1	Verificabilidad	Alta
Descripción	El sistema usará en los distintos lenguajes estructuras para capturar excepciones		

ID: RS-23	TIPO: Sistema	NECESIDAD: Deseable	
Nombre	Tiempo de respuesta	ID Req. Usuario	RU-12
		Fuente	Consultor
Iteración	1	Verificabilidad	Alta
Descripción	El sistema no tolerará una diferencia mayor a 10 segundos entre la captura de un evento y su replicación en la cloud		

5.3. Taxonomía de Eventos Android relacionados con malware

En este apartado, como consecuencia del análisis en profundidad realizado de la taxonomía incluida en [1], se realiza una clasificación a un alto nivel de los principales datos de un terminal Android que son potencialmente susceptibles de ser usados/monitorizados por una aplicación maliciosa. Así como los elementos Android en los que se traducen y que son susceptibles de encontrar en la implementación del código malware. La clasificación se realiza en base a elementos del dispositivo que tienen rasgos en común, y que como se puede observar, la mayoría guardan mucha relación con los datos comentados en la sección del estado del arte, en concreto en el estudio de las aplicaciones maliciosas.

Los principales elementos son:

- Recursos de Comunicación.
- Sensores
- Recursos Hardware
- Recursos de Usuario
- Almacenamiento
- Otros datos relevantes

5.3.1. Recursos de Comunicación

Cualquier sistema de comunicación es muy susceptible de ser utilizado por el malware, ya que permite el contacto del dispositivo con el exterior. Se utiliza para acciones básicas relacionadas

con el malware, como medio de fuga de información, envío o interceptación de SMSs, aunque también se realizan operaciones más sensibles como para que el propio malware contacte con un C&C, permitiendo el control remoto el dispositivo.

5.3.1.1. Llamadas

Un malware puede realizar llamadas o incluso interceptarlas para obtener el número de teléfono. El objetivo es obtener información personal o bien realizar llamadas a números determinados y obtener beneficio económico. Los principales permisos que utiliza un malware con respecto a llamadas son:

- android.permission.CALL_PRIVILEGED
- android.permission.PROCESS_OUTGOING_CALLS
- android.permission.CALL_PHONE

Además de la implementación de los métodos propios que otorgan este uso: onCallStateChanged(). Esto método se sobrescribe para detectar información relevante cuando se generan llamadas entrantes o salientes.

5.3.1.2. Mensajes

En este apartado los recursos más importantes son: lectura de mensajes (datos personales), recepción de mensajes (lectura de número) y envío de mensajes (supone gasto económico). Los permisos más utilizados en la monitorización de mensajes son:

- android.permission.READ_SMS
- android.permission.RECEIVE_SMS
- android.permission.SEND_SMS
- android.permission.WRITE_SMS

En análisis estático el método más importante que se debe buscar es: sendTextMessage().

5.3.1.3. Red

Permite la comunicación con el exterior y el envío de información. Las aplicaciones malware que realizan fuga de información a través de red utilizan principalmente los siguientes permisos:

- android.permission.INTERNET
- android.permission.ACCESS_NETWORK_STATE
- android.permission.CHANGE_NETWORK_STATE
- android.permission.CHANGE_WIFI_STATE
- android.permission.ACCESS_WIFI_STATE

5.3.2. Sensores

Se enumeran en una tabla los principales permisos relacionados con sensores que un malware puede solicitar en el AndroidManifest.

Cámara – Fotografía o Vídeo	• android.permission.CAMERA
GPS	• android.permission.ACCESS_COARSE_LOCATION • android.permission.ACCESS_FINE_LOCATION
Movimiento (métodos)	• getSensorList() • getDefaultSensor()

5.3.3. Recursos Hardware

5.3.3.1. Batería

El malware suele monitorizar el nivel de la batería, así como de si se encuentra el dispositivo conectado a la corriente. Es potencialmente sensible a ser monitorizado por malware que pertenezcan a botnets. Se identifica a través de un *BroadcastReceiver* específico para los cambios de la batería, del cual se pueden obtener datos: android.intent.action.BATTERY_CHANGED.

5.3.3.2. Eventos de teclado

Se considera relevante tanto cuando se pulsa una tecla, como cuando se suelta, e incluso cuando se pulsa de manera prolongada. Se pueden capturar estos eventos cuando se sobrescriben los métodos con el código que el desarrollador desea en cada caso. Para ello se busca en el código fuente de las clases .java los métodos: onKeyUp(), onKeyDown() y onKeyLongPressed().

5.3.3.3. Información del dispositivo

El malware tiende a consultar el IMEI del dispositivo, el modelo, estado de la red, operador de red e incluso el país donde se encuentra. Para acceder a estos datos se necesita declarar el permiso android.permission.READ_PHONE_STATE, y lo más crítico a lo que se puede acceder se obtiene capturando el uso de los siguientes métodos: getDeviceId(), getNetworkOperatorName(), getCellLocation().

5.3.4. Recursos de Usuario

5.3.4.1. Aplicaciones de Terceros y Activities

Una aplicación malware puede consultar la lista de aplicaciones que hay instaladas en el dispositivo así como el nombre de todas las Activities que hay. Con ello existe el riesgo de gestionar los paquetes instalados e incluso de borrarlos. Además la aplicación podría descargar una aplicación externa e instalarla de manera silenciosa. Esto se monitoriza controlando el uso de los permisos:

- android.permission.INSTALL_PACKAGES
- android.permission.DELETE_PACKAGES

5.3.4.2. Contactos

Existe el riesgo de que una aplicación lea los contactos (id, nombre, telefono1, telefono1, etc). Se debe controlar el permiso: android.permission.READ_CONTACTS.

5.3.4.3. Cuentas de usuario

Se debe monitorizar el uso del método getAccounts(), así como el permiso que se requiere: android.permission.GET_ACCOUNTS.

5.3.5. Almacenamiento

Como método principal en la lectura de datos sensibles por el malware previo a fuga del mismo, se deben monitorizar los accesos de lectura de la tarjeta SD. También es importante comprobar los accesos de lectura/escritura a bases de datos SQLite.

5.3.6. Otros datos relevantes

Es relativamente importante si el malware tiene permisos para recibir el *Broadcast* emitido por el terminal cuando termina de iniciar. El permiso es: android.permission.RECEIVE_BOOT_COMPLETED, y la acción recibida: android.intent.action.BOOT_COMPLETED.

Por otro lado, como es lógico en un sistema basado en el kernel de Linux, es crítica la posibilidad del malware de ganar acceso de super-usuario. Para ello se debe comprobar la ejecución del AndroidRunTime, y en concreto ejecutar el comando “su” (*super user*).

5.4. Eventos generados por la acción del Usuario

El estudio realizado sobre las acciones de usuario sirve para mejorar el sistema y dar más funcionalidad ampliando el comportamiento del terminal que se puede replicar en la cloud. Las acciones más importantes identificadas son la gestión de paquetes, como instalación y desinstalación de aplicaciones, el lanzamiento de actividades, etc.

Como se ha comentado anteriormente, el desarrollo del proyecto es un proceso iterativo, por lo que el catálogo de acciones de usuario está compuesto por dos versiones. La versión inicial tiene las posibles acciones que se identificaron en un primer momento, de manera muy ambiciosa se estudió qué puede realizar un usuario en un terminal, y qué evento genera esa acción en el sistema, para posteriormente ver si es factible de alguna manera replicar en la nube la acción.

Después de diseñar el sistema e implementar la detección y replicación de algunas acciones se comprobó que no todo era factible, por lo que se hizo un segundo análisis donde queda finalmente mostrado la relación de evento – acción que se incorpora al sistema. También se muestran las acciones que se identificaron en el momento inicial y que no han sido factibles abordar, bien por limitación del sistema o del entorno que no permite la replicación.

5.4.1. Tabla Evento-Acción Inicial

Se muestra un resumen del excel generado de la relación eventos – acciones que se propone como análisis inicial de las posibles acciones que puede realizar un usuario con un dispositivo Android. Se adjunta la tabla entera en el apartado 10 anexos.

Nombre	Evento	Acción	Comentario
Encender modo vuelo	[D/AirplaneModeEnabler(314): onAirplaneModeChanged : airplaneModeEnabled true]	[sqlite3 /data/data/com.android.providers.settings/databases/settings.db "insert into system values(null, 'airplane_mode_on', 1);"] [am broadcast -a android.intent.action.SERVICE_STATE]	Al emitir el broadcast se finaliza el proceso com.android.phone de manera inesperada. Para quitar el mensaje del mismo, se insertan 2 eventos de teclado, un movimiento a la izqda para situar el foco en al boton cerrar, y un enter para cerrar la ventana | adb -e shell am broadcast -a android.intent.action.AIRPLANE_MODE --ez state true
Apagar modo vuelo	[D/AirplaneModeEnabler(320): onAirplaneModeChanged : airplaneModeEnabled false]	[sqlite3 /data/data/com.android.providers.settings/databases/settings.db "insert into system values(null, 'airplane_mode_on', 0);"] [am broadcast -a android.intent.action.SERVICE_STATE]	–
Desactivar 3G	D/ConnectivityService(128): setMobileDataEnabled(false)	sqlite3 /data/data/com.android.providers.settings/databases/settings.db "insert into secure values(null, 'mobile_data', 0);" [am broadcast -a android.intent.action.SERVICE_STATE]	svc data dis/enabled | telnet gsm data off
Establecer llamada	I/ActivityManager(128): Starting: Intent { act=android.intent.action.CALL_PRIVILEGED dat=tel:xxx-xxx-xxxx flg=0x10000000 cmp=com.android.phone/.PrivilegedOutgoingCallBroadcaster } from pid 1437	–	–
	I/am_new_intent(61):[1080501888,14,com.android.phone/.InCallScreen,android.intent.action.CALL,NULL,tel:1-234-567-89,281018368]	adb -e shell am start -a android.intent.action.CALL_PRIVILEGED -d tel:1-234-567-89	–
Recibir llamada	I/PhoneUtils(632): getCallerName : 123456789	telnet gsm call "numero entrante"	–
Aceptar llamada recibida	I/phone (632): acceptCall: incoming...	telnet gsm accept "numero entrante"	–
Activar 3G	D/ConnectivityService(128): setMobileDataEnabled(true)	sqlite3 /data/data/com.android.providers.settings/databases/settings.db "insert into secure values(null, 'mobile_data', 1);" [am broadcast -a android.intent.action.SERVICE_STATE]	No funciona, ni svc, ni telnet. Deshabilitan el icono pero no restringe la conexion de datos.

Tabla - 9 - Evento - Acción Inicial

5.4.2. Tabla Evento-Acción Final

El resultado y las conclusiones a las que se han llegado después de estudiar el diseño y la implementación del Excel anterior se muestran a continuación, con la relación evento – acción que se ha implementado en el sistema. Además se adjunta la relación de acciones identificadas inicialmente pero que no se han tenido en cuenta, así como su justificación. La tabla completa se adjunta como anexo.

Nombre	Evento	Acción	Comentario
Abrir archivo	_	_	Pendiente de Implementar
Abrir Carpeta	_	_	Pendiente de Implementar
Aceptar llamada recibida	CaptureEvents (Event Listener)	Telnet	_
Activar 3G	Evento de Sistema (ConnectivityService)	Broadcast en Apk Replicador	_
Añadir/eliminar Contacto	CaptureEvents (Content Observer)	_	Pendiente de Implementar
Apagar GPS	Evento de Sistema (GpsLocationProvider)	Broadcast en Apk Replicador	_
Apagar modo vuelo	Evento de Sistema (AirplaneModeEnabler)	Broadcast en Apk Replicador	_
Apagar Terminal	CaptureEvents (Event Listener)	Nativo (adb)	_
Bloquear pantalla	CaptureEvents (BroadcastRegistrado)	Nativo (adb)	Se envia el evento Tecla Power
Cambiar estado a cargando	CaptureEvents (BroadcastRegistrado)	Telnet	_
Cambiar estado de carga de batería	CaptureEvents (BroadcastRegistrado)	Telnet	_
Cambiar fondo de pantalla	CaptureEvents (BroadcastReceiver)	Broadcast en Apk Replicador	Implementado en APK
Cambiar Hora sistema	CaptureEvents (BroadcastReceiver)	Broadcast en Apk Replicador	Valido igualmente para cambio de fecha, como de hora
Cancelar / Colgar Llamada	CaptureEvents (Event Listener)	Telnet	_
Conectar AC	CaptureEvents (BroadcastRegistrado)	Telnet	_
Desactivar 3G	Evento de Sistema (ConnectivityService)	Broadcast en Apk Replicador	_
Desconectar de AC	CaptureEvents (BroadcastRegistrado)	Telnet	_

Tabla - 10 -Evento - Acción Final

Nombre	Evento	Acción	Comentario
Activar Wifi	D/WifiService(128): setWifiEnabled: true	Sin Acción	No se puede replicar en emulador
Desactivar Wifi	D/WifiService(128): setWifiEnabled: false	Sin Acción	No se puede replicar en emulador
Montar SD	Evento de Sistema (Vold)	Sin Acción	No se puede replicar si la aplicación no tiene permisos de system
Desmontar SD	Evento de Sistema (MountService)	Sin Acción	No se puede replicar si la aplicación no tiene permisos de system
Borrar/formatear SD	Evento de Sistema (MountService)	Sin Acción	No se puede replicar si la aplicación no tiene permisos de system
Factory Reset	Evento de Sistema (MasterClear)	Sin Acción	necesario establecer APK como administrador de dispositivos, y añadir política de wipedata
Bluetooth Encender	Evento de Sistema (BluetoothService)	Broadcast en Apk Replicador	No se puede replicar en emulador
Bluetooth Apagar	Evento de Sistema (BluetoothService)	Broadcast en Apk Replicador	No se puede replicar en emulador
Boton atrás pulsar	Evento de Sistema (ScreenCaptureAction)	Sin Acción	No implementado (adb -e shell input keyevent 4)
boton home	Evento de Sistema (WindowManager)	Sin Acción	No implementado (adb -e shell input keyevent 3)
boton menu	Evento de Sistema (SettingsProvider)	Sin Acción	No implementado (adb -e shell input keyevent 82)
boton power		Sin Acción	No implementado (adb -e shell input keyevent 26)
Desconectar de usb	Evento de Sistema (notified_event)	Sin acción	No se puede replicar
Conectar USB	Evento de Sistema (Setting)	Sin Acción	No se puede replicar
Gestionar Cuentas	Sin evento	Sin Acción	Sin implementar
Modificar Teclados / idiomas	Sin evento	Sin Acción	Sin implementar
Apagar hora Automatica	Evento de Sistema (SettingsProvider)	Sin Acción	Sin implementar

Tabla - 11 - Evento - Acción Final - Acciones no contempladas

5.5. Procesos de Sistema

El sistema desarrollado no tiene actores ni usuarios más que la persona que interactúa con el terminal para generar acciones. Es un sistema innovador basado en la investigación y el análisis cuya finalidad es replicar el comportamiento de un terminal generado por la interacción de un usuario y las aplicaciones malware.

Los procesos del sistema están claramente definidos, pero todo constituye un proceso automatizado basado en el alcance de la implementación del sistema, que se corresponde con la cantidad de eventos que se capturan, y la cantidad de acciones que se replican. El alcance final del sistema en cuanto a captura de eventos y replicación de acciones está definido por el excel del apartado anterior (Anexo 3).

La arquitectura a que se ha llegado después de dos iteraciones en el desarrollo del sistema se explica a continuación en forma de procesos del sistema.

5.5.1. Captura de eventos

Para poder analizar y replicar el comportamiento del terminal se debe capturar de alguna manera. La forma más básica es con la información que el propio sistema operativo Android proporciona a través de logs o registros del mismo.

Los eventos [13] que se generan en Android están agrupados en 3 log buffers:

- Main.
- Events
- Radio
- System

Estos archivos se actualizan en tiempo real y están ubicados en la ruta: /dev/log/. Se muestra a continuación el esquema general del sistema de gestión de eventos de Android.

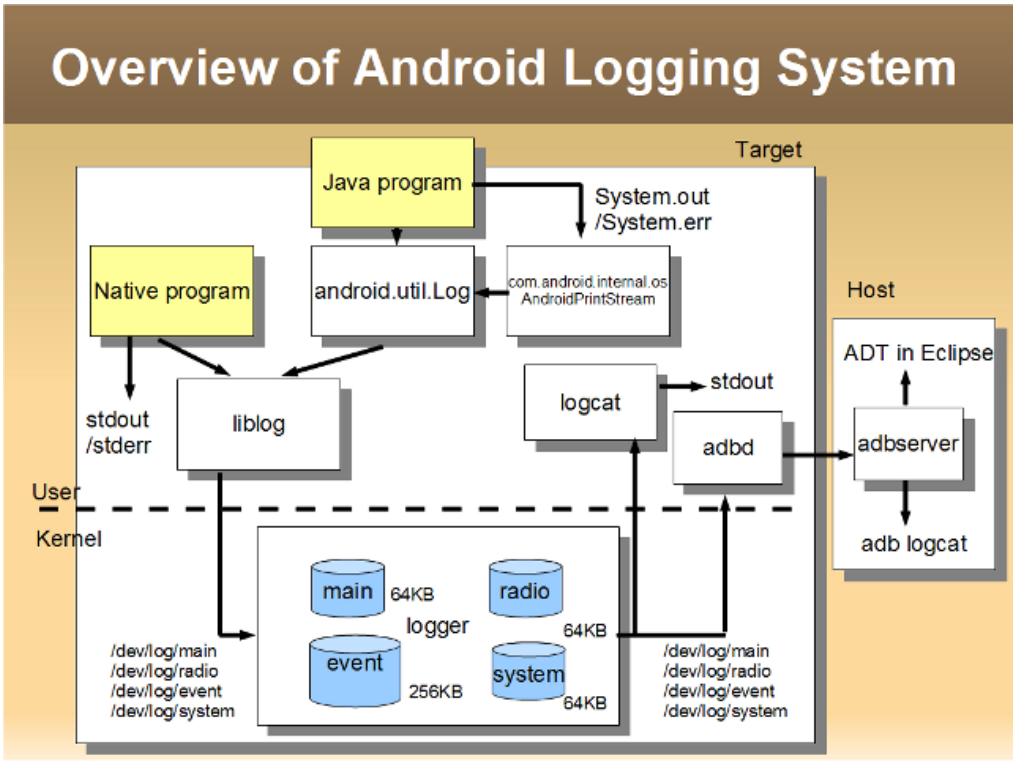


Figura - 30 - Android Logging System [13]

Como se observa en la imagen los eventos son generados por programas nativos o aplicaciones Android (API de Java) utilizando la librería liblog para generar los eventos. Se pueden leer a través de la salida estándar utilizando logcat.

Se explica la estructura de los eventos con el siguiente ejemplo:

"D/PackageManager(527): Removing package com.estrongs.android.pop"

- "***D***" es el nivel de detalle, prioridad o verbosity (Debug, Error, Info, Verbose, Warn).
- "***PackageManager***" es el TAG utilizado, título del evento, o como lo vamos a llamar en este sistema, tipo de evento.
- "***(527)***" es el ID del proceso que ha generado el evento, no tiene relevancia en absoluto para nuestro sistema.
- "***: Removing package com.estrongs.android.pop***" es el mensaje del evento. Es la parte que se va a *parsear* para obtener la acción a replicar. Cada tipo de evento puede tener en el mensaje distintas acciones.

En el análisis realizado sobre los eventos generados por acción de usuario se ha comprobado que hay eventos importantes que no se envían al logcat para poder ser capturados. Esto ha obligado a implementar una aplicación Android e instalarla en el terminal Android que detecte las acciones que no se envían por defecto al logcat, y que cree un evento propio (de tipo "*CAPTUREVENTS*") que sí se pueda capturar.

De esta manera también se consigue independizar la versión de Android del terminal físico (en los casos donde es posible), ya que dependiendo de la versión, los eventos de sistema para una

misma acción pueden tener una estructura distinta en el mensaje del evento, e incluso en el tipo de evento. Si la aplicación captura el evento a través de la API de Android, funciona para cualquier versión por lo que se genera un mismo evento. Con ello se consigue también mejorar la flexibilidad del sistema ya que no hay que realizar un parseo en el sistema por evento distinto que es generado por distintas versiones para una misma acción.

5.5.2. Parseo de Inputs

En el caso de los eventos propios, la estructura es la comentada, el tipo de evento o TAG es "*CAPTUREVENTS*". El mensaje enviado depende de cada acción, pero siempre está formado por una subetiqueta que indica el tipo de acción, y después los parámetros. Todo ello separado por el carácter barra vertical. Dos ejemplos son los siguientes:

- "*D/CAPTUREVENTS(440): TAGBATTERY | LEVEL:50 | PLUGGED:1 | STATUS:2"*
- "*D/CAPTUREVENTS(433): TAGCALL | OUTGOING CALL: 12345123*"

Los eventos de sistema no tienen una estructura determinada, cada tipo de evento tiene uno o varios mensajes que hay que *parsear* de distinta manera, distinguiendo la acción para ser generada. El parseo de estos eventos se hace en base al estudio de los propios eventos que se desean monitorizar, y adaptando en cada caso el filtrado de objetos dentro del mensaje. Un ejemplo de un tipo de evento con varias acciones es y proveniente de distintas versiones de Android es:

"*I/ActivityManager(1108): START u0 {act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] flg=0x10200000 cmp=com.android.browser/.BrowserActivity bnds=[192,385][240,433]} from pid 1430*"

En este evento la forma de comienzo del mensaje START u0 es característica de la versión 4.2.2 y la acción es abrir el navegador por defecto de Internet:

"*I/ActivityManager(1108): Starting: Intent { dat=file:///mnt/sdcard/SinMachismoSi.apk cmp=com.android.packageinstaller/.InstallAppProgress (has extras) } from pid 785*"

Este otro evento del mismo tipo es generado en un terminal con versión 2.3.3 y la acción es instalar una aplicación. Es un ejemplo de distintas acciones en un mismo evento, que deben ser *parseadas* de distinta manera.

5.5.3. Replicación de acciones en la cloud

Dependiendo del tipo de evento, la acción que se quiera realizar y el análisis realizado sobre cómo se pueden replicar las acciones estudiadas se debe utilizar o bien una herramienta externa, o bien enviar un *Broadcast* al cloud para activar la acción en la aplicación propia instalada.

La idea de implementar una aplicación e instalarla en la cloud surge de la limitación que existe en la replicación de ciertas acciones de manera externa, Android y sus aplicaciones de sistema tienen sus procesos internos y no se pueden replicar todas las líneas de código de manera externa. Existen programas nativos de Android como "*svc*" que permite gestionar la comunicación, pero tan solo a nivel interno, es decir, por ejemplo si se desactiva la red de datos con "*svc*" se ha comprobado que no se puede navegar con el navegador, pero el icono de 3G de la pestaña notificaciones no desaparece. Así surge la idea de que sea una aplicación interna la que realice acciones en la cloud, a través de códigos enviados desde fuera.

La comunicación está basada en la emisión de *Broadcast* con una serie de extras que definen la acción a ejecutar según el evento *parseado*. La aplicación lee los extras y en base a ellos ejecuta la sección adecuada del código que replica la acción en la cloud.

Las herramientas que permiten acciones en la cloud desde el terminal son:

- **Comando ADB**. Permite finalizar el cloud, instalar y desinstalar aplicaciones, entre otras acciones genéricas.
- **Telnet**. Se puede establecer una conexión telnet con la cloud que permite modificar estadísticas de la batería y sensores entre otros.
- **MonkeyRunner**. Esta funcionalidad la proporciona el Android SDK y permite comenzar actividades junto con los parámetros necesarios como componentes, datos o flags.

La utilización de todas estas herramientas ha surgido por análisis e iteración en el sistema.

Los procesos por tanto quedan alineados de la siguiente manera:

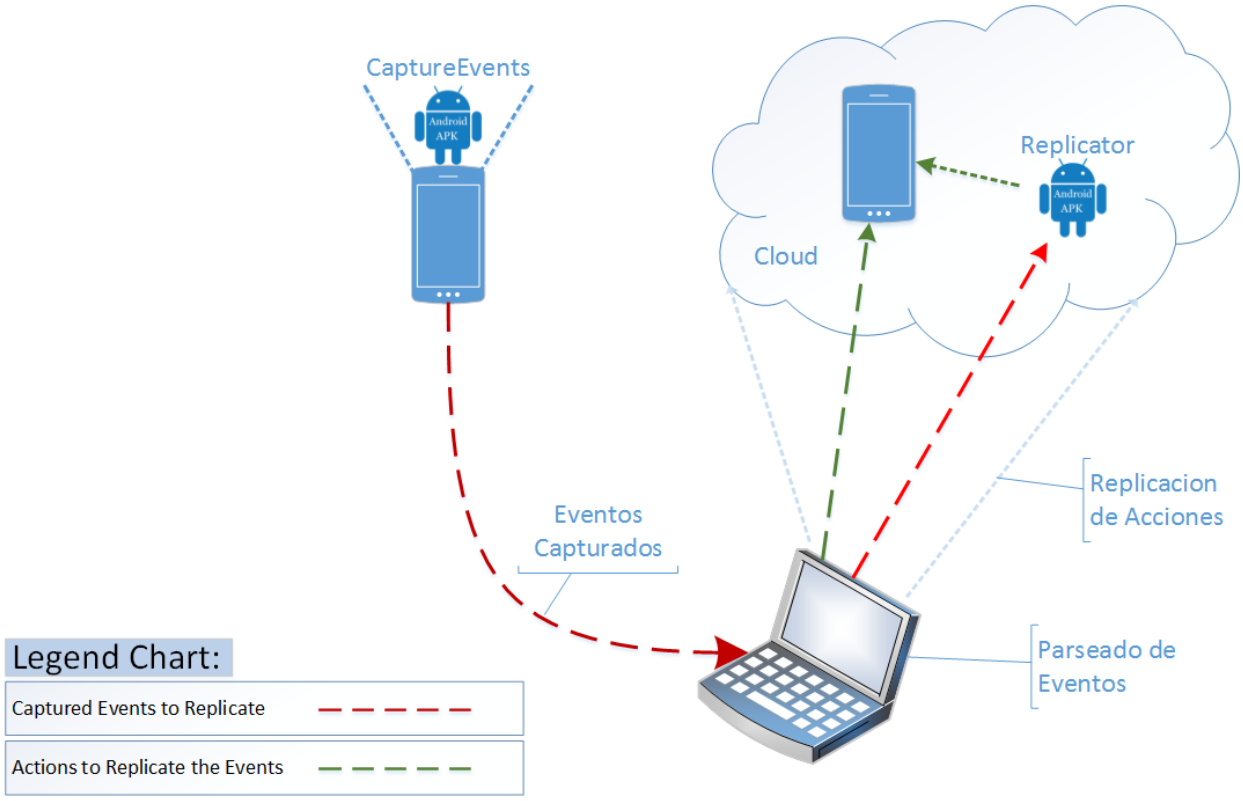


Figura - 31 - Estructura de Replicación de Acciones

5.6. Entorno operativo y herramientas a utilizar

5.6.1. CosecCloneCloud

Como se adelantó en el Estudio de Viabilidad esta herramienta va a ser la base del sistema, por lo que se continúa su desarrollo y amplía su funcionalidad para satisfacer los requisitos del sistema.

Este sistema en sí está programado en Python además de varios scripts en bash shell que permiten la preparación y comunicación del sistema con el terminal físico.

La funcionalidad del sistema está programada en dos archivos principales:

1. device2clone.py. Recibe por entrada estándar los eventos raw que genera el terminal móvil y después de decodificar el evento genera un objeto Event (programado en el segundo archivo) que *parsea* el mensaje del evento y si está contemplado en el sistema genera la acción correspondiente al mismo. Esta acción se ejecuta con la llamada de un método.
2. LogcatEvent.py. Contiene la estructura principal del sistema, es donde se amplía la funcionalidad de las acciones que se van a replicar en función de los eventos de entrada.

5.6.2. Droidbox

La integración de Droidbox en la cloud es trivial. Cuando se levanta el cloud (Android Virtual Device) utilizando un script, se realiza modificando los parámetros de la máquina virtual Android seleccionando la imagen del sistema y el estado de la RAM específica de Droidbox, que contiene los mecanismos de monitorización de aplicaciones malware.

Inicialmente el sistema Droidbox permitía monitorizar las aplicaciones en Android Virtual Devices de la versión 2.3.3. Actualmente la comunidad ha ampliado el sistema pudiendo utilizar Droidbox con la versión 4.1.

Esta limitación supone un problema para el funcionamiento ideal de nuestro sistema, ya que el objetivo es garantizar la funcionalidad independientemente de la versión Android. El desarrollo de este sistema para nuevas versiones, al igual que la adquisición de un terminal físico de cada versión queda fuera de lugar en el alcance actual del proyecto.

Aun sin poder tener el sistema ideal, las acciones que se replican están basadas en la API de Android de manera genérica, por lo que el sistema funciona igual utilizando como Cloud la versión 2.3.3 de Droidbox.

5.6.3. Elementos Android

A continuación se exponen las herramientas y utilidades de la API de Android en las que se basan las aplicaciones que forman parte del sistema.

5.6.3.1. Activity

Las actividades de Android son interfaces o pantallas que representan de manera visual las aplicaciones que permiten la interacción del usuario con el terminal. Una aplicación puede tener gran cantidad de actividades, teniendo distinta funcionalidad en cada una de ellas. Cada vez que se lanza una actividad se muestra una interfaz nueva cuya funcionalidad está definida por la clase Java que hereda de Activity, y por el archivo .xml que organiza los objetos visuales de la interfaz.

5.6.3.2. Intent

Existen acciones predefinidas en un terminal Android, como por ejemplo realizar una búsqueda a través del GPS pasando unas coordenadas, establecer una alarma en el dispositivo, realizar una fotografía a través de la cámara. Estas acciones son reutilizables por lo que no es necesario programarlas cada vez que se quiera utilizar, basta con comenzar un Intent que implemente esta actividad. Generalmente están relacionadas con componentes del sistema.

En el proyecto se implementa la escucha de este tipo de acciones para determinar el comportamiento de Android desde el interior.

5.6.3.3. Services

El concepto de servicio en Android es idéntico a los servicios que existen en otros entornos. Los servicios son procesos que se ejecutan en background, sin una interfaz visual y sin la interacción

de un usuario. Tienen una mayor prioridad que las actividades, por lo que resultan ideales para realizar tareas de monitorización y escucha de información.

En la aplicación instalada en el terminal Android que captura eventos se implementa un servicio que está a la escucha de que ocurran determinados eventos en el sistema, para poder analizarlos y que sean enviados al sistema de registro de eventos.

5.6.3.4. ContentObserver

Antes de definir ContentObserver y la necesidad de implementar este sistema de monitorización de cambios en Android, se debe definir qué es ContentProvider.

Un proveedor de contenidos en Android es un sistema que de manera estructurada almacena información de una aplicación y permite una interfaz de gestión de los datos almacenados. Es un punto intermedio entre la programación de la aplicación y el almacenamiento interno que se realiza en bases de datos SQLite. En definitiva, el almacén de los datos se realiza en una base de datos propia de cada aplicación pero su acceso y gestión se realiza a través de la API de ContentProvider.

ContentObserver es una clase de Android que permite la recepción de notificaciones cuando un determinado ContentProvider es modificado. Permite por tanto registrar los cambios en el almacenamiento de información que deseemos para determinar qué actualizaciones se han realizado.

En la aplicación CaptureEvents se implementan en concreto dos listener de ContentProviders:

1. **SMS**. Cuando se realizan cambios en el contenedor de SMS, bien sea por enviar un sms, guardar un borrador, recibir, eliminar, mover, etc., se recibe una notificación. Sobre la notificación actualmente está implementado el registro de envío de SMS, por lo que se genera un evento propio con el número destinatario y el cuerpo del mensaje cuando se envía al exterior.
2. **Contactos**. Cuando existe alguna modificación en los contactos del terminal, se recibe la notificación.

5.6.3.5. BroadcastReceiver

BroadcastReceiver es una clase de Android que cuando se hereda permite recibir notificaciones al producirse ciertos cambios en el sistema. Es utilizado para escuchar mensajes generados por el sistema al ocurrir eventos importantes, o bien para recibir notificaciones cuando ocurran Intents específicos.

Es utilizado en la aplicación CaptureEvents para ciertos comportamientos del sistema, y por la aplicación instalada en la cloud para recibir un Intent propio emitido desde el exterior, y poder así replicar la acción definida en los extras del *Broadcast*.

Sección 6

DISEÑO

6.1. Introducción al Diseño

Una vez analizado el sistema y establecido con gran de detalle qué se desea construir, se debe realizar para cada módulo del sistema un estudio de diferentes alternativas sobre las que comenzar a elaborar el sistema cumpliendo con los requerimientos establecidos.

Primero se va a realizar un análisis de la integración de los módulos, cuya arquitectura supone el diseño final que tiene el sistema desarrollado.

Posteriormente se va a definir cada uno de los módulos principales analizando las diferentes alternativas de arquitectura mostrando el modelo de clases final. Para finalizar se expone un diagrama del ciclo de vida de un evento en el sistema desde que es generado en el terminal hasta que se replica en la cloud.

Como se ha comentado anteriormente, el sistema es un proceso automatizado por lo que no se almacena información ni se requiere realizar un modelo de datos del mismo.

6.2. Integración de procesos

En este apartado se explica cómo los distintos procesos se comunican entre sí para transferir los eventos desde la captura hasta que son replicados en la cloud, y como queda finalmente el diseño del sistema.

Existen tres posibilidades para iniciar la ejecución del sistema, que son gestionadas por el lanzador “device_launcher” escrito en Python. Permite iniciar la capturación de eventos propios del terminal, y establecer la conexión para que el sistema reciba como inputs los eventos.

Cuando se reciben los eventos, primero se decodifican para evitar que caracteres especiales (suelen aparecer muchos dependiendo del sistema y versión de Android) produzcan excepciones y paradas en el sistema. El evento decodificado sirve como entrada para generar un objeto de tipo Evento (si está su clase implementada), que *parsea* el mensaje del evento y realiza la acción de replicación.

Si no se puede realizar la acción de manera externa, se manda un mensaje a la aplicación instalada en la cloud, que realice la acción.

El sistema final generado posee la siguiente arquitectura:

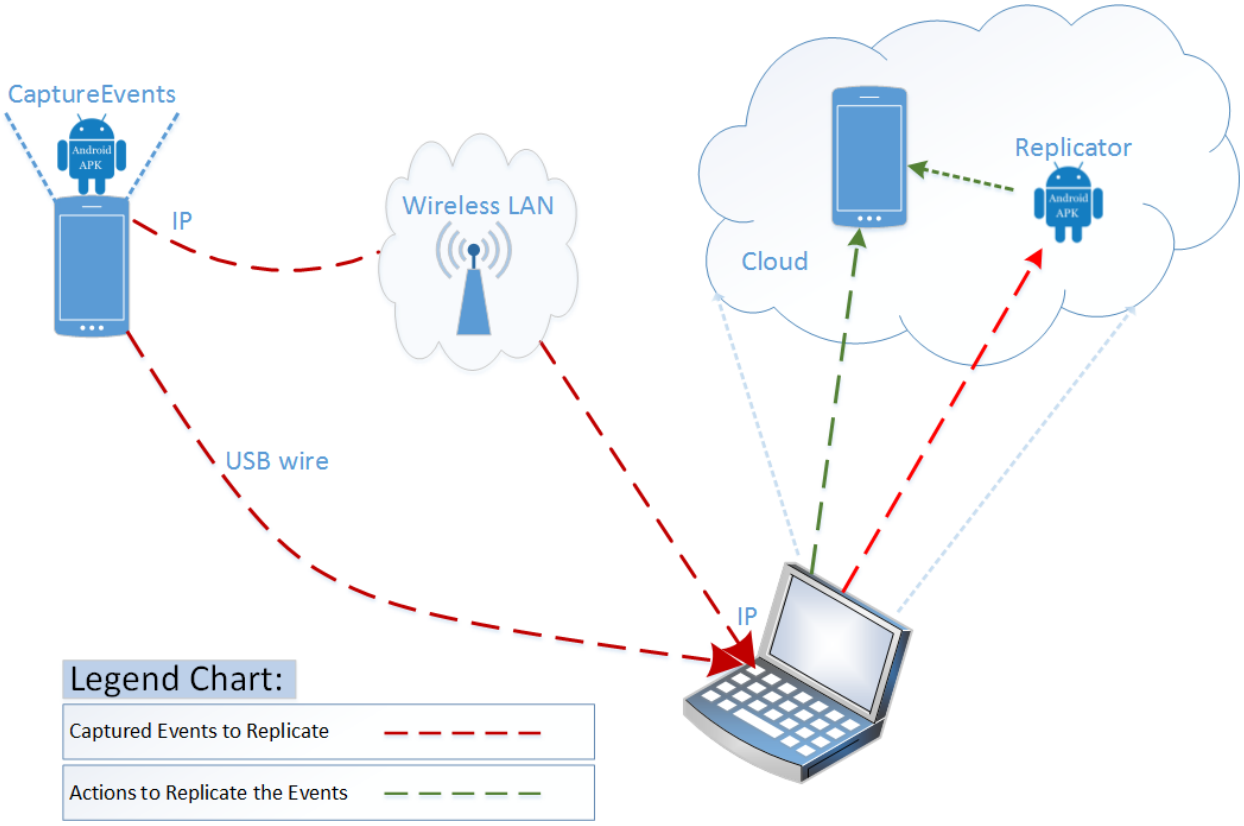


Figura - 32 - Arquitectura de Sistema

En el terminal se produce la captura de eventos, bien sean los generados por el sistema, como los propios que se han generado a través de la aplicación. Estos eventos son recopilados en el servidor para ser *parseados* y convertidos en objetos de tipos Evento. Estos objetos se ejecutan replicando la acción correspondiente a la cloud, o bien se notifica una acción a través de *Broadcast* a la aplicación Replicator, instalada en la cloud.

6.3. Diseño Captura de Eventos

6.3.1. Arquitectura Captura de Eventos

Dentro de la arquitectura vamos a definir dos fases, aunque la segunda afecta más bien a la integración de los eventos en el sistema.

La arquitectura inicial estaba basada únicamente en la recolección de los eventos que son generados por el sistema. Como hemos visto en el análisis esto impedía capturar el comportamiento determinado. No todo se replica automáticamente por el sistema como nos gustaría. La arquitectura inicial es:

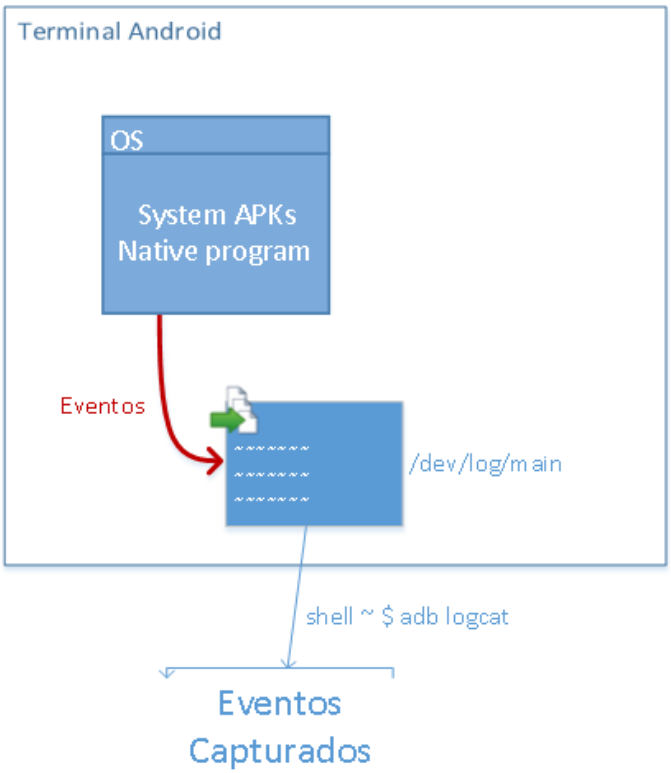


Figura - 33 - Arquitectura inicial de captura de eventos

Por ello se ha creado un sistema adicional que hace uso de la API de Log de Android, que con mecanismos de escucha y detección se pueden generar eventos propios en el sistema.

La aplicación está desarrollada en Java con el entorno de desarrollo de Android (SDK). El nombre del proyecto es: "*CaptureEvents*" y el paquete: "*es.uc3m.cosec.captureevents*". Para que la aplicación sea compatible con todas las versiones Android se ha definido como API mínima donde va a funcionar la 10 (versión 2.3.3) y como versión objetivo (para la cual la aplicación está optimizada) la 19 (4.4).

```
<uses-sdk
    android:minSdkVersion="10"
    android:targetSdkVersion="19" />
```

Los permisos que requiere la aplicación para poder acceder a la información que se desea monitorizar son:

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.PROCESS_OUTGOING_CALLS" />
<uses-permission android:name="android.permission.RECEIVE_SMS" />
<uses-permission android:name="android.permission.READ_SMS" />
<uses-permission android:name="android.permission.READ_CONTACTS" />

<uses-permission android:name="android.permission.CAMERA" />
```

Los *BroadcastReceivers* que declara la aplicación para obtener información sobre las acciones son:

```
<action android:name="android.intent.action.BOOT_COMPLETED" />
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
        <action android:name="android.provider.Telephony.SMS_RECEIVED" />
        <action android:name="android.provider.Telephony.SMS_DELIVER" />
        <action android:name="android.media.RINGER_MODE_CHANGED" />
        <action android:name="android.media.VOLUME_CHANGED_ACTION" />
        <action android:name="android.intent.action.TIME_SET" />
        <action android:name="android.intent.action.TIMEZONE_CHANGED" />
        <action android:name="android.hardware.action.NEW_PICTURE" />
        <action android:name="com.android.camera.NEW_PICTURE" />
        <action android:name="android.intent.action.ACTION_SHUTDOWN" />
        <action android:name="android.intent.action.WALLPAPER_CHANGED" />
```

La arquitectura resultante es por tanto:

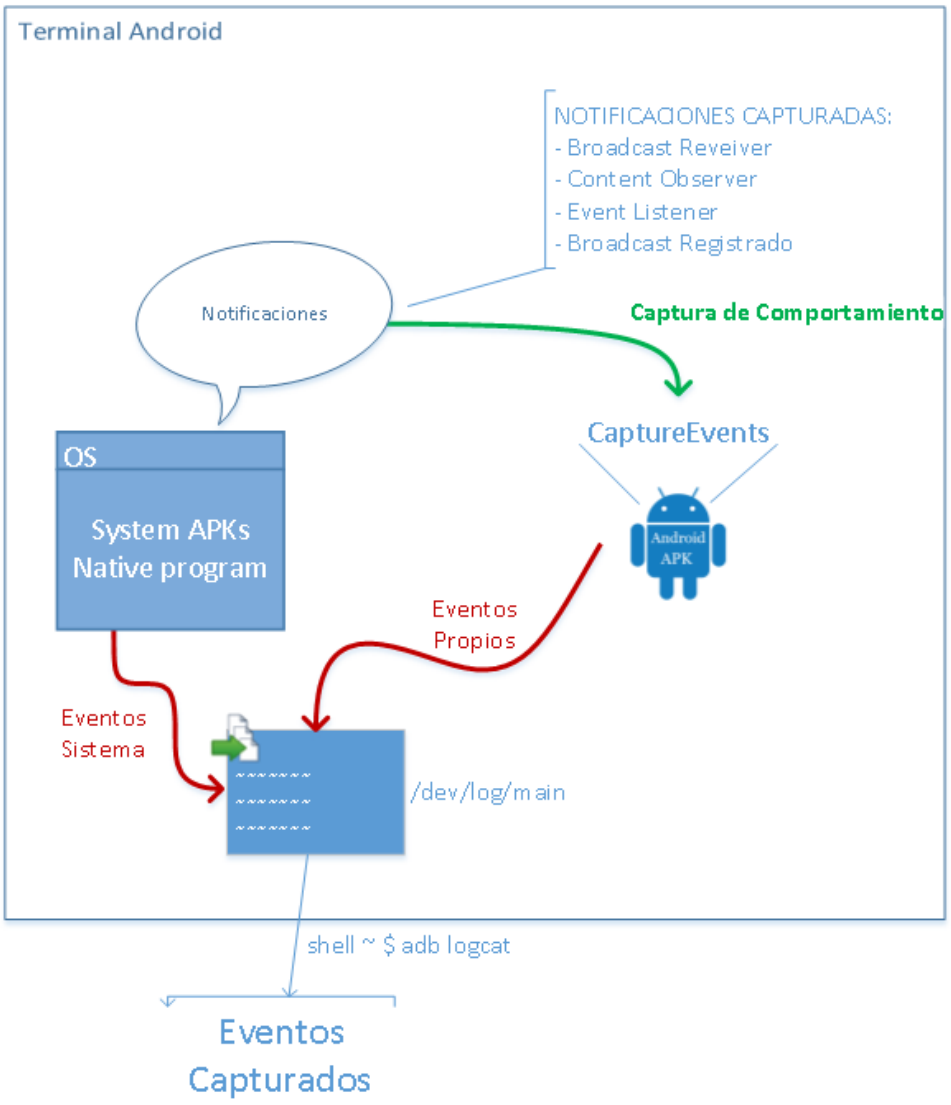


Figura - 34 - Arquitectura final de captura de eventos

6.3.2. Diagrama de Clases CaptureEvents

A continuación se muestra como queda el diagrama de clases de la aplicación Android CaptureEvents. Cada clase tiene una funcionalidad específica de captura de eventos, salvo ServiceActivity, actividad que sirve como puente para iniciar el servicio de la aplicación.

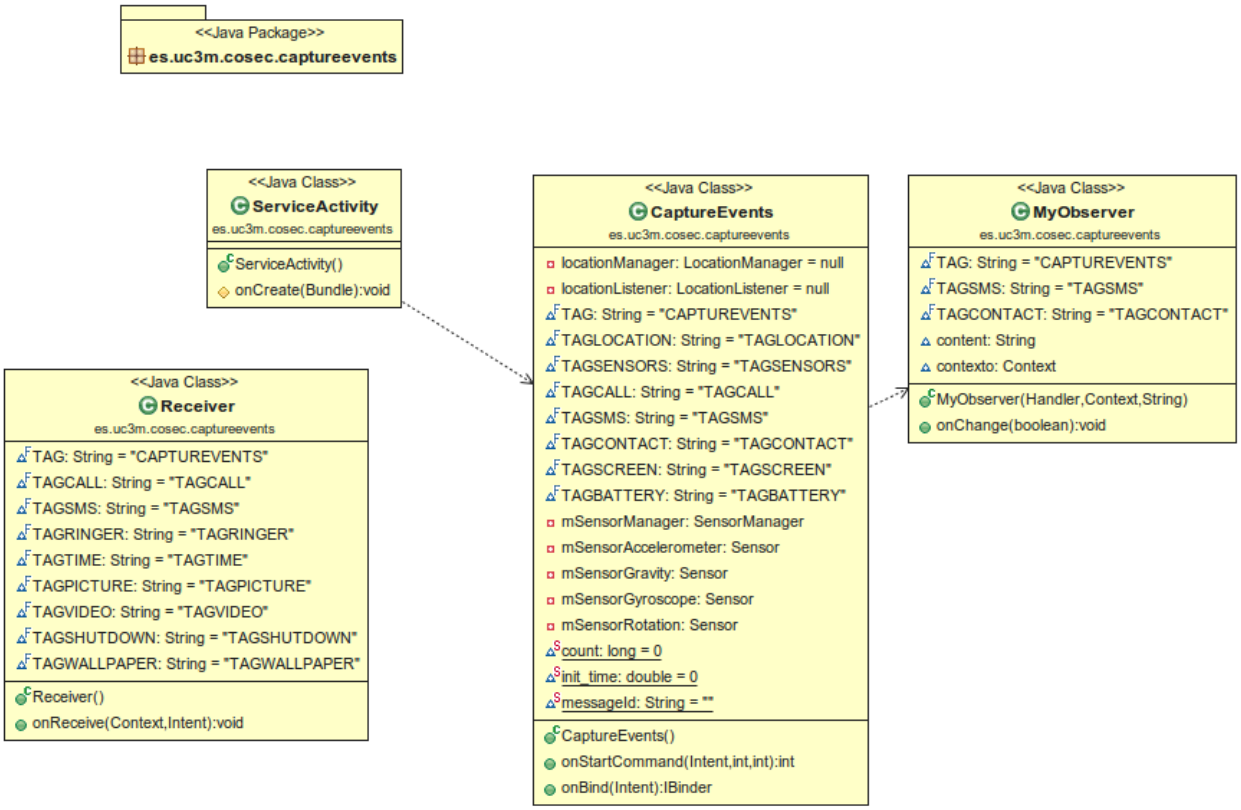


Figura - 35 - Diagrama de Clases de CaptureEvents

6.4. Diseño Sistema CosecCloneCloud

6.4.1. Arquitectura CosecCloneCloud

La arquitectura de este módulo viene dado por defecto, no se ha mejorado más que con la integración de una clase para integrar la ejecución comandos Telnet y *Broadcast*, además de la ampliación de las clases que replican las acciones de los eventos.

La arquitectura inicial es:

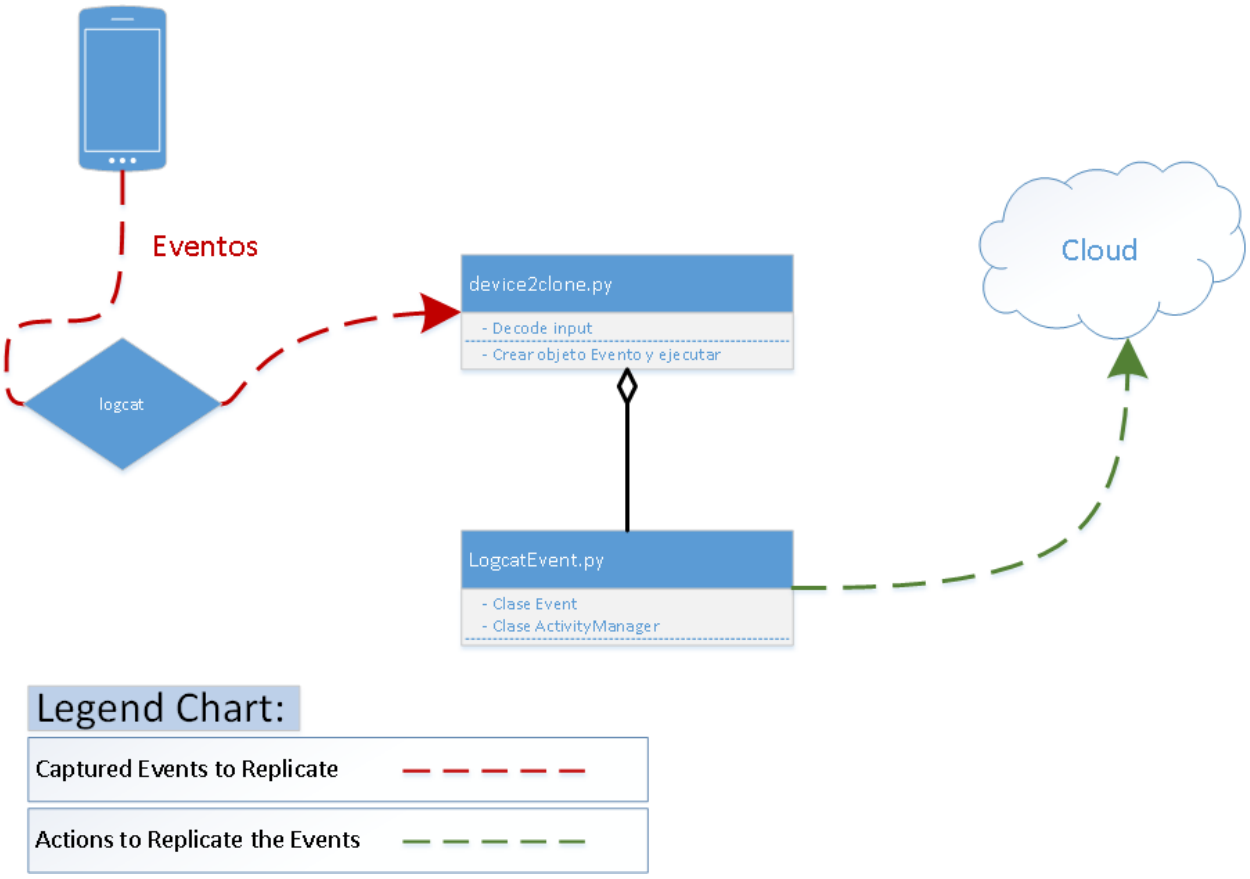


Figura - 36 - Arquitectura inicial CosecCloneCloud

Los eventos del terminal son recogidos únicamente a través de logcat por USB. Únicamente se replican los eventos de tipo ActivityManager.

Con la ampliación realizada la arquitectura queda de la siguiente manera:

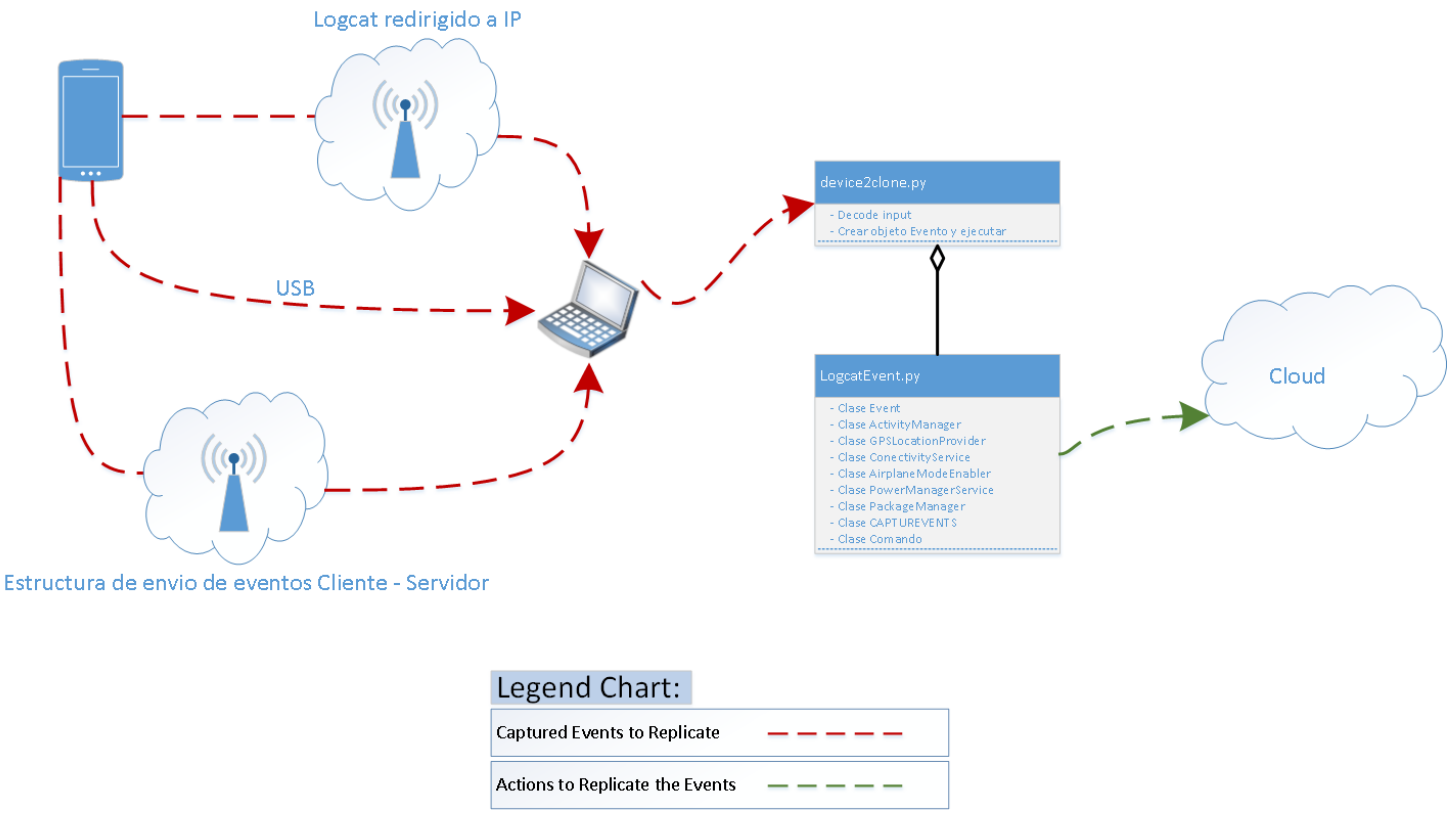


Figura - 37 - Arquitectura final CosecCloneCloud

Se ha ampliado la arquitectura para que el sistema funcione a través de WiFi. En concreto se han diseñado dos soluciones distintas a este requerimiento.

1. Se redirige la captura de eventos a través de Logcat a una IP, por lo que si el sistema está conectado a una red, independientemente de la que sea, envía los eventos a la IP de destino.
2. Se ha diseñado una estructura cliente-servidor con aplicaciones nativas programadas en C. La primera instalada en el terminal se considera el cliente que envía a una IP o nombre de máquina los eventos leídos de manera interna de los registros. En el lado del servidor hay una aplicación también escrita en C que abre un socket y permite la escucha de los eventos que se reciben del terminal.

6.4.2. Diagrama de clases

El estado inicial del sistema CosecCloneCloud permite la comunicación e integración de los otros dos sistemas debido a los scripts que ejecutan logcat y establecen la conexión tanto con el terminal físico como con la cloud. El centro de la funcionalidad se encuentra en el archivo "*LogcatEvent.py*", donde existe una template base para reutilizar en cada evento nuevo que se desee implementar. Por ello se muestra a continuación el diagrama de clases de este archivo, en su estado inicial.

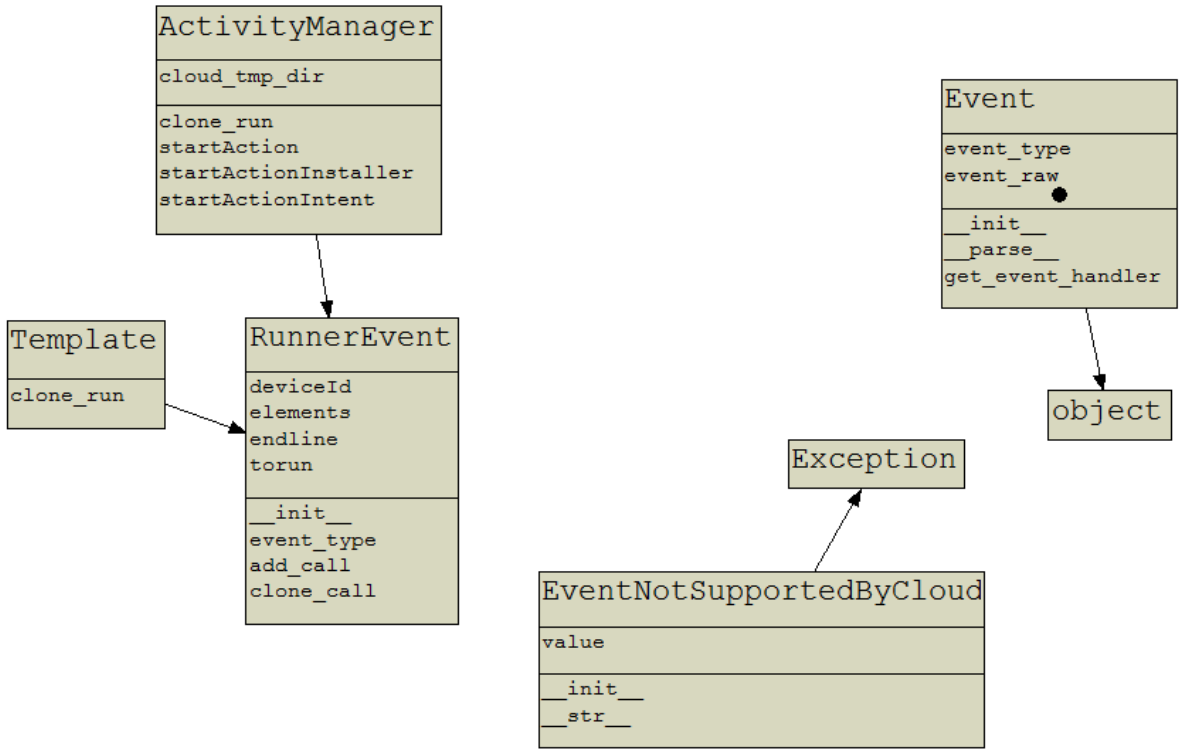


Figura - 38 - Diagrama de Clases inicial de CosecCloneCloud

Un objeto de tipo Event almacena la información relacionada con cada evento input del sistema. En concreto se crea el tipo de evento: "*ActivityManager*" por ejemplo. El método handler permite obtener una instancia de la clase implementada de dicho método para ejecutar las acciones que sean oportunas. Cada una de estas clases implementan RunnerEvent, un sistema que gestiona a alto nivel las llamadas de las que consta la acción, para después ejecutar la acción con clone_run.

En el siguiente diagrama se muestra el diseño final de las clases implementadas en CosecCloneCloud:

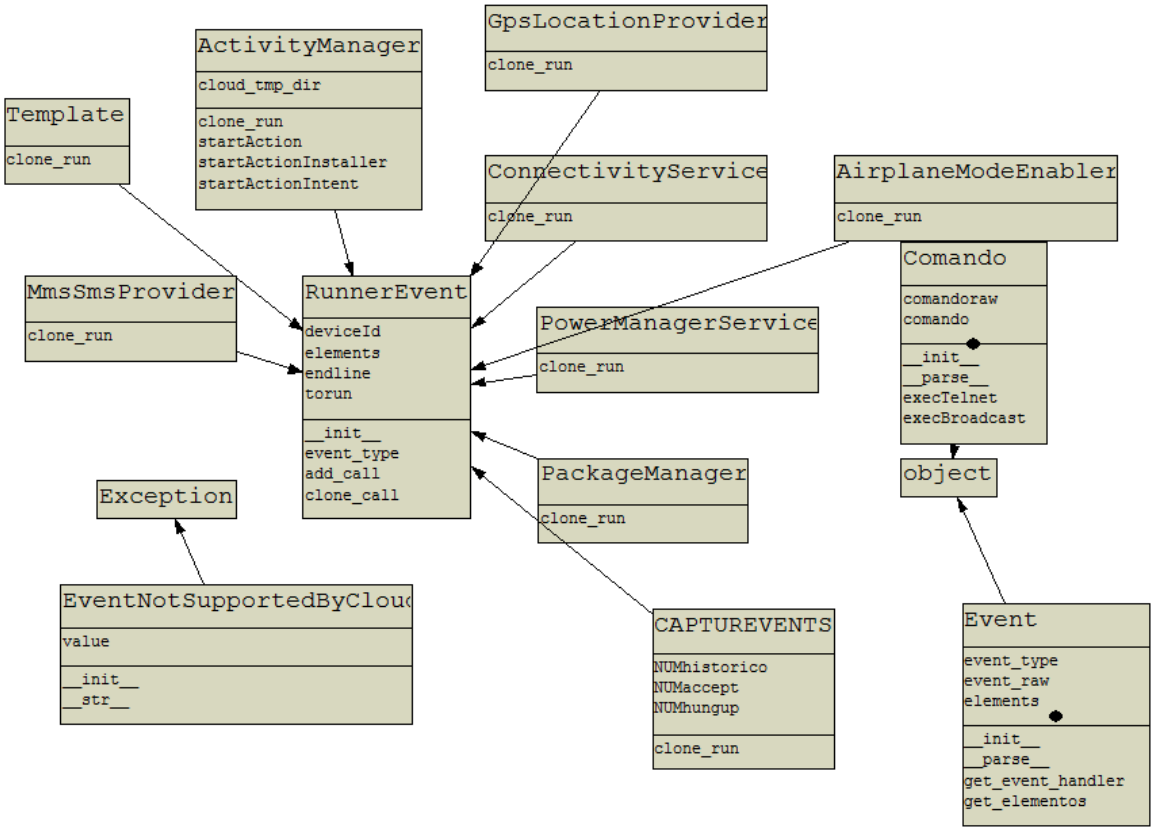


Figura - 39 - Diagrama de Clases final de CosecCloneCloud

Las mejoras más significativas han sido la inclusión de una clase que gestione distintos tipos de acciones específicas, como son utilizar un comando Telnet en la Cloud, y realizar un *Broadcast* específico para generar las acciones que de otra manera no se pueden. La otra mejora significativa es la implementación de la clase "*CAPTUREVENTS*", que gestiona los eventos capturados de este tipo y permite la ejecución de la acción correspondiente al evento.

Por otro lado, se observa el resto de clases implementadas que se corresponden con distintos eventos de sistema.

6.5. Diseño Replicador de Eventos

Esta actividad ("*Replicator*") surge como necesidad de no poder replicar manera externa todos los eventos. La arquitectura en el sistema es la siguiente:

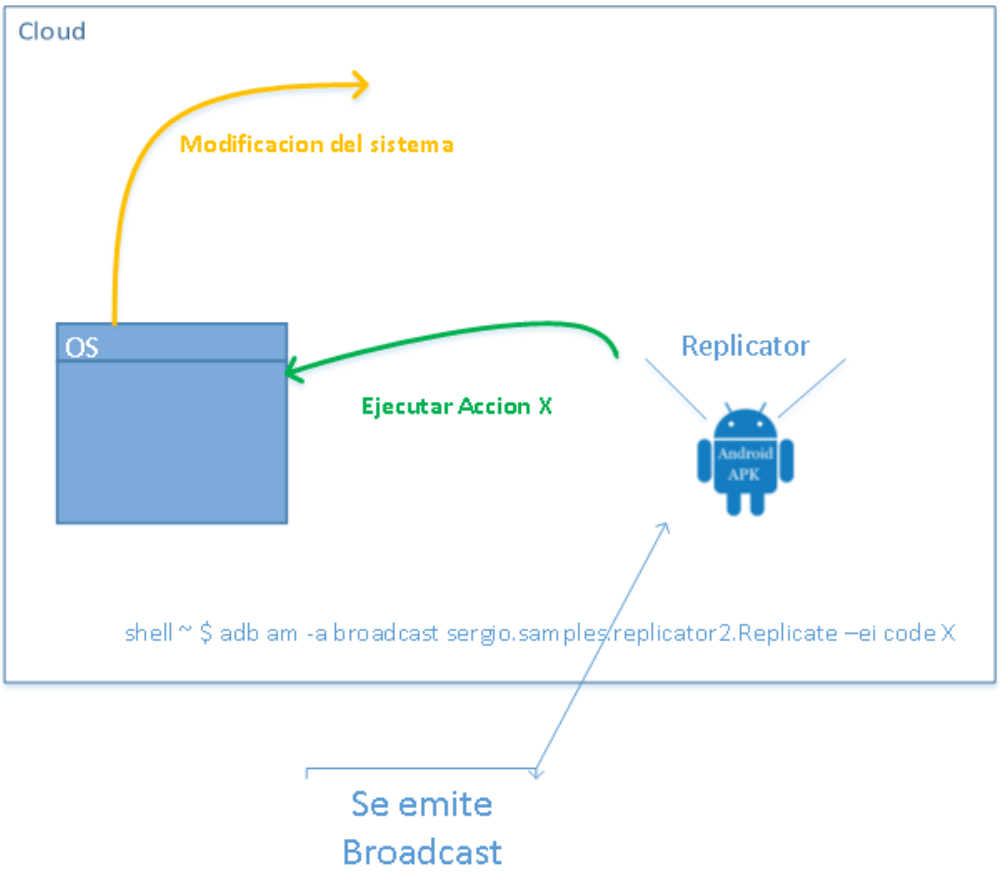


Figura - 40 - Diseño aplicacion Replicator

Se ha generado una estructura que permite la comunicación entre el exterior y la aplicación. El *Broadcast* emitido se explica a continuación con un ejemplo:

```
// code = 1 | Air Plane Mode
// @extra Boolean {true,false}

"adb -e shell am broadcast -a es.uc3m.cosec.replicator.REPLICATE --ei code 1 --ez enabled false"
```

Este *Broadcast* de código 1 indica una modificación del modo avión en la cloud. El extra pasado como parámetro "*enabled*" de tipo boolean indica a la aplicación Replicator que debe apagar el modo avión.

6.5.1. Diagrama de clases Replicator

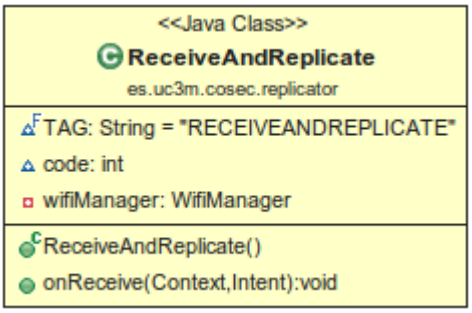


Figura - 41 - Diagrama de Clases Replicator

Se trata de una única clase que hereda de *BroadcastReceiver* por lo que sobrescribe el método onReceive(), que escucha el Intent enviado desde fuera ("*es.uc3m.cosec.replicator.REPLICATE*") y dependiendo del código pasado replica una acción u otra en el sistema.

6.6. Alternativas Cloud

6.6.1. Androidx86

El primer Cloud que se tuvo en cuenta está basado en la emulación de una máquina virtual del sistema android-x86 sobre la que instalar las aplicaciones malware a monitorizar. Se podría conseguir una replicación de acciones de usuarios pero no el uso de Droidbox, ya que el desarrollo de android-x86 se basa en un entorno completo de sistema operativo. Droidbox está implementado para integrar su sistema de imágenes modificadas con un Android Virtual Device.

Las ventajas que se obtendrían con este diseño serían principalmente rapidez y fluidez en la transición de eventos y replicación de acciones, además de poder guardar y almacenar snapshots concretas del estado de la VM.

Los principales inconvenientes es la falta de flexibilidad, no hay imágenes de sistemas operativos android-x86 de todas las APIs ni se podría adaptar al entorno de monitorización Droidbox, por lo que supone una restricción principal.

6.6.2. Android Virtual Device

Android Virtual Device es una utilidad que posee el entorno de desarrollo de Android (Android SDK) que se proporciona al público para desarrollar y testear aplicaciones. En concreto este sistema emula un dispositivo virtual en el PC a través del software de emulación QEMU. Permite la emulación de elementos hardware de los dispositivos físicos y permite modificar imágenes del sistema modificadas.

El AVD manager permite gestionar los dispositivos virtuales de Android en el equipo. Se pueden emular dispositivos de cualquier versión de Android, ya que a través del SDK se pueden instalar y desinstalar actualizaciones de los sistemas Android disponibles, así como otros elementos de drivers, documentación, etc.

Además de porque esta alternativa viene integrada con CosecCloneCloud, se ha convertido en el cloud o sandbox ideal al integrarse con Droidbox, y ser el método más utilizado por otros desarrolladores para emular dispositivos virtuales y testear aplicaciones.

El ejecutable de Android SDK "*emulator*" es el que posee la capacidad para generar y modificar los dispositivos virtuales de Android en el sistema. A través de parámetros y opciones a la hora de levantar el cloud se carga la imagen de Droidbox para añadir funcionalidad al sistema de monitorización de aplicaciones.

Como inconveniente se puede destacar que para realizar la monitorización con Droidbox, se debe de utilizar únicamente la versión 2.3.3 de Android en el dispositivo virtual. Esto es debido a que la implementación de la funcionalidad de Droidbox radica en la modificación de la imagen del sistema de un Android 2.3.3, y no otra versión.

En principio no afecta a la operativa del sistema, ya que la replicación de acciones en la cloud no depende de la versión de la propia cloud, es decir, las acciones capturadas en un terminal físico con versión 4.4 por ejemplo, van a ser replicadas en la cloud con versión 2.3.3.

6.7. Integración de Procesos

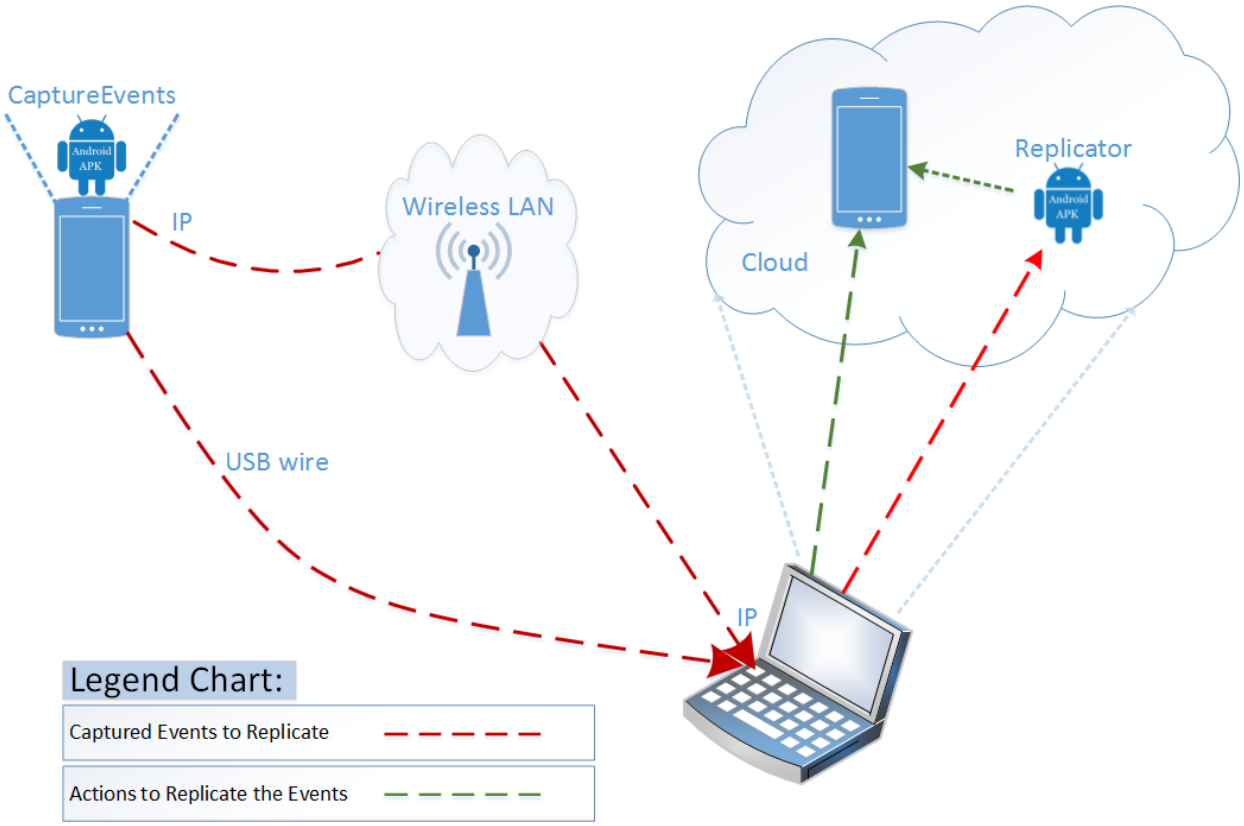
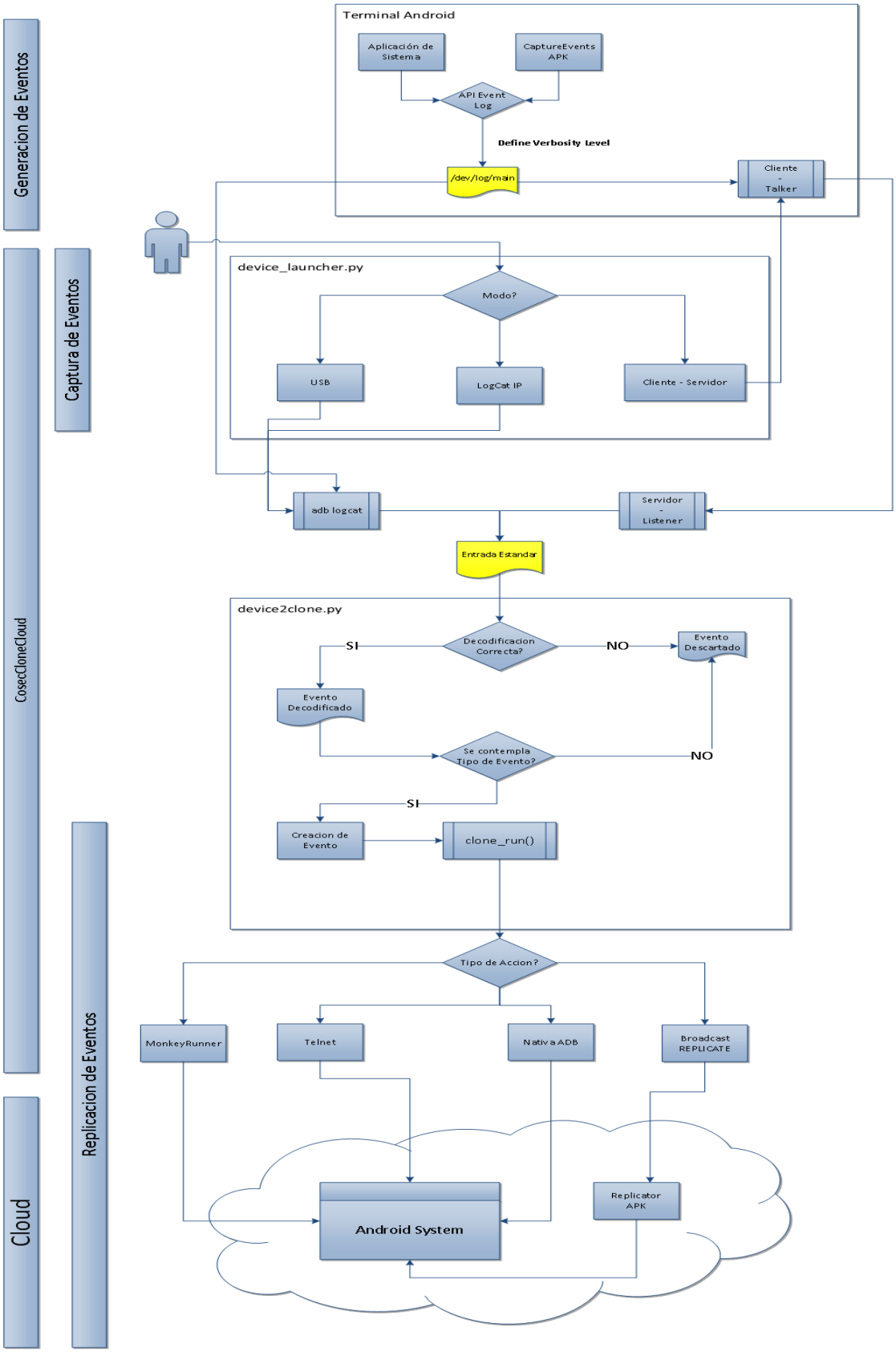


Figura - 42 - Arquitectura de Sistema

6.8. Ciclo de vida del Sistema

Para facilitar la visión del sistema y conocer en profundidad qué tratamiento sufre el evento durante todo el ciclo de vida, se va a realizar un diagrama explicativo con las principales situaciones que se pueden llegar a producir en el mismo.

Figura - 43 - Ciclo de Vida del Sistema


Generacion de Eventos
Captura de Eventos
CosecCloneCloud
Replicacion de Eventos
Cloud
Terminal Android
Aplicación de Sistema
CaptureEvents APK
API Event Log
Define Verbosity Level
/dev/log/main
Cliente - Talker
device_launcher.py
Modo?
USB
LogCat IP
Cliente - Servidor
adb logcat
Servidor - Listener
Entrada Estandar
device2clone.py
Decodificacion Correcta?
SI
NO
Evento Descartado
Evento Decodificado
Se contempla Tipo de Evento?
NO
SI
Creacion de Evento
clone_run()
Tipo de Accion?
MonkeyRunner
Telnet
Nativa ADB
Broadcast REPLICATE
Android System
Replicator APK

A continuación se explica cada fase del ciclo de vida de un evento.

1. Generación de Eventos.

Los eventos son almacenados en el log desde dos fuentes principales. El sistema y sus aplicaciones que forman los eventos de sistema, y por otro lado los eventos propios que tienen la etiqueta “CAPTUREVENTS” y son generados por la aplicación CaptureEvents.

2. Captura de Eventos.

Dependiendo de la ejecución que decida hacer el usuario, los eventos serán capturados de distinta manera. Si se selecciona a través de USB, serán replicados utilizando logcat. Si se selecciona de manera inalámbrica, se realizará a través de logcat, pero escuchando en la ip del dispositivo físico. Por último utilizando la estructura cliente-servidor, donde primero se ejecuta el listener en el servidor, que escucha los eventos que envía el cliente, cuando se ejecuta en el terminal el programa.

3. Parseo de Eventos.

Los eventos escuchados son pasados a través de la entrada estándar al programa “device2clone.py” que primero decodifica el evento raw, y si no hay problemas, comprueba si está contemplado ese evento, es decir, si existe una clase que implemente acciones sobre ese tipo de evento. Si está contemplado se crea un objeto del tipo que corresponda y se ejecuta su acción.

4. Ejecución de Acción

La acción puede ejecutarse de manera externa utilizando: la herramienta Monkey de Android, la API de Telnet para Python, que permite emitir eventos al AVD y de manera nativa utilizando la funcionalidad de ADB.

Por otro lado, si la acción no se puede realizar de manera externa, se emite un *Broadcast* a la nube que es capturado por la aplicación “Replicator”.

5. Replicación de la Acción en la Cloud.

Si la acción viene como *Broadcast*, Replicator ejecuta la acción pasada como parámetro en el *Broadcast*. De esta manera conseguimos la replicación en la nube del evento capturado en el terminal físico.

Sección 7

IMPLEMENTACIÓN

7.1. Introducción

En este apartado se va a explicar con detalle la implementación de cada clase, y como se integra con el sistema para permitir la comunicación entre los distintos procesos. En los casos donde se identifiquen excepciones o partes del código poco amigables al entendimiento sencillo, así como los problemas encontrados, se justificará y explicará con detenimiento. La finalidad de este apartado es proporcionar una visión asequible y sencilla de cómo funcionan los principales procesos del sistema y la integración de los mismos para generar el ciclo de vida de un evento en su replicación.

Para comenzar, se muestra a continuación el directorio raíz del proyecto junto con los archivos que contiene. Para facilitar la visualización se ha incluido a modo de leyenda la segregación por colores del estado inicial del sistema y del final que se ha desarrollado. Las carpetas/archivos en contraste naranja -- se corresponde con lo nuevo implementado en el sistema, mientras que lo marcado en verde -- se corresponde con los elementos modificados durante el desarrollo del proyecto. El resto no se ha modificado.

```
/root
        /Cliente-Servidor
        |       android_talker2.c
        |       listener.c
        /CaptureEvents
        |       AndroidManifest.xml
        |       /src/es/uc3m/cosec/captureevents
                |       CaptureEvents.java
                |       MyObserver.java
                |       Receiver.java
                |       ServiceActivity.java
        /Replicator
        |       AndroidManifest.xml
        |       /src/es/uc3m/cosec/replicator
                |       ReceiveAndReplicate.java
        /cosecclonecloud
        |       clonerunner.py
        |       cloud2behaviour.py
        |       cloud_monitoring.py
        |       device2clone.py
        |       device_launcher.py
        |       device_monitoring.py
        |       device_monitoring_wifi_2.sh
        |       killingAdb.py
        |       startupcloud.sh
```

```
|       /device2cloneEvents
|           LogcatEvent.py
|       /droidbox
|           cosec_droidbox.py
|           cosec_monkeyrunner.py
|       /gen_clonerunner
|       /imagesDroidbox
|           ramdisk.img
|           system.img
|           userdata.img
|           zImage
|       /tmp_data
|       /utils
|           Replicator.apk
|           CaptureEvents.apk
|           killProcess.sh
|           ev.sh
|           listener
|           andro_talker2
```

7.2. Implementación de CaptureEvents

A continuación se detalla la implementación de la aplicación CaptureEvents. Esta aplicación es reinstalada en el terminal físico cuando se inicia el sistema con la ejecución de device_launcher.py.

- **CaptureEvents.java**

Esta clase es la principal del capturador de eventos propios del sistema. Es un servicio ya que hereda de la clase Service. La funcionalidad principal que tiene es registrar listeners de distintos elementos que proporcionan información relevante frente al comportamiento que se pretende capturar, ya sea por ser acción de usuario o por ser elementos sensibles de monitorizar por malware. Es obligatorio implementar esta clase como servicio ya que al iniciarlo se registran los listeners que se quedan a la espera de recibir información. La generación de eventos propia se hace en base al siguiente comando:

```
Log.d(TAG , TAGLOCATION + " | Lat:" + location.getLatitude() + " | Long:" +
```

En este ejemplo se observa la estructura. TAG es un literal definido como variable global cuyo valor es siempre: "*CAPTUREVENTS*" para poder identificar con facilidad este tipo de eventos. A continuación se incluye una SUBTAG que indica el tipo de evento propio capturado, el valor de este tipo de etiquetas coincide exactamente con el nombre de la variable.

Después de la subtag se incluyen siempre barras horizontales para separar el mensaje del evento, que servirá como información para generar la acción durante la replicación.

Otro ejemplo de captura de evento de esta clase es la interceptación de llamadas entrantes al dispositivo:

```
PhoneStateListener mPhoneListener = new PhoneStateListener() {
public void onCallStateChanged (int state, String incomingNumber) {
      try {
         switch (state) {
         case TelephonyManager.CALL_STATE_RINGING:
            Log.d(TAG , TAGCALL + " | RINGING: " + incomingNumber);
            break;
         case TelephonyManager.CALL_STATE_OFFHOOK:
            Log.d(TAG , TAGCALL + " | OFFHOOK: " + incomingNumber);
            break;
         case TelephonyManager.CALL_STATE_IDLE:
            Log.d(TAG , TAGCALL + " | IDLE: " + incomingNumber);
            break;
         default:
             Log.d(TAG , TAGCALL + " | Unknown phone state ");
         }
      } catch (Exception e) {
            Log.d(TAG , TAGCALL + " | phone state listener EXCEPTION");
      }
   }
};
```

- **MyObserver.java**

Como se ha explicado durante el diseño, esta clase escucha los cambios en los ContentProviders sobre los cuales se ha registrado un ContentObserver como es esta clase. El registro se realiza en la clase CaptureEvents para poder ponerse a escuchar cuando se levanta el servicio.

La única funcionalidad actual de esta clase es la captura del evento entrada de SMS, a pesar de estar implementada parte de la escucha de modificación de contactos del teléfono.

Para escuchar los cambios producidos se registra en el servicio:

```
// listen to sended SMS
MyObserver observer = new MyObserver(new Handler(), this, android.provider.Telephony.Sms.CONTENT_URI.toString());
contentResolver.registerContentObserver(android.provider.Telephony.Sms.CONTENT_URI,true, observer);
```

La implementación del método onChange es de sobreescritura por herencia de clase. Para capturar la acción de envío de SMS se realiza:

```
@Override
public void onChange(boolean selfChange) {
        super.onChange(selfChange);
        Cursor cur;
        if (android.provider.Telephony.Sms.CONTENT_URI.toString().compareTo(content)==0) {
                try{
                cur = contexto.getContentResolver().query(android.provider.Telephony.Sms.CONTENT_URI, null, null, null, null);
                        if (cur.moveToFirst()) {
                                String body = cur.getString(
                 cur.getColumnIndexOrThrow("body")).toString();
                                String address = cur.getString(
                 cur.getColumnIndexOrThrow("address")).toString();
                                if(cur.getString(cur.getColumnIndexOrThrow("type")).toString().compareTo("4") == 0){
                        Log.d(TAG , TAGSMS + " | OUTGOING SMS | number: " + address + " | body: " + body.replace('\n', ' '));
                                }
                        }
                }catch(Exception e){
                        Log.d(TAG , TAGSMS + " EXCEPTION " + e.getMessage() + " | " + e.toString());
                }
        } else if (... CONTENT = CONTACTS ...)
```

- **Receiver.java**

Esta clase al heredar de *BroadcastReceiver* puede recibir las notificaciones de los *BroadcastReceiver* a los cuales está suscrito. Una suscripción de *BroadcastReceiver* no es más que declarar lo siguiente en el AndroidManifest:

```
<receiver
   android:name=".Receiver"
   android:enabled="true"
   android:exported="true"
   android:permission="android.permission.BROADCAST_SMS" >
   <intent-filter>
      <action android:name="android.intent.action.BOOT_COMPLETED" />
      <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
      <action android:name="android.provider.Telephony.SMS_RECEIVED" />
      <action android:name="android.provider.Telephony.SMS_DELIVER" />
      <action android:name="android.media.RINGER_MODE_CHANGED" />
      <action android:name="android.media.VOLUME_CHANGED_ACTION" />
      <action android:name="android.intent.action.TIME_SET" />
      <action android:name="android.intent.action.TIMEZONE_CHANGED" />
      <action android:name="android.hardware.action.NEW_PICTURE" />
      <action android:name="com.android.camera.NEW_PICTURE" />
      <action android:name="android.intent.action.ACTION_SHUTDOWN" />
      <action android:name="android.intent.action.WALLPAPER_CHANGED" />
      <action android:name=" es.uc3m.cosec.captureevents.BOOT" />
   </intent-filter>
</receiver>
```

En este caso, nuestro receiver está suscrito a tantos *Broadcast* para detectar todas estas acciones y generar el evento propio correspondiente.

La implementación de la clase se basa en un switch incluido en el método onReceive(). Cada case coincide con un *Broadcast* de lo declarado. La acción en cada uno de ellos es incluir de nuevo el registro del evento con la etiqueta correspondiente, por ejemplo:

```
switch(intent.getAction()){
case "android.intent.action.NEW_OUTGOING_CALL":
        // log outgoing call with number
        String number = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);
        Log.d(TAG , TAGCALL + " | OUTGOING CALL: " + number);
        break;
```

- **ServiceActivity.java**

Esta Actividad es ejecutada por device_launcher.py cuando se inicia la ejecución del sistema únicamente para levantar el servicio CaptureEvents y comenzar la escucha de las acciones definidas. Este método no es para nada ortodoxo, pero solventa muchos problemas relacionados con permisos cuando previamente se intentaba levantar el servicio desde el exterior con:

“am startservice -n es.uc3m.cosec.captureevents/.CaptureEvents”

Por tanto, el único método de esta actividad que es donde se lanza el servicio, se implementa de la siguiente manera:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Intent myIntent = new Intent(this, CaptureEvents.class);
        this.startService(myIntent);
        this.finish();
}
```

7.3. Implementación de CosecCloneCloud

Este apartado se centra en dos elementos muy importantes, primero los scripts que permiten la comunicación entre los distintos procesos del sistema, y por otro lado la funcionalidad principal de parseo de eventos y replicación de acciones en la cloud.

- **device_launcher.py**

Se trata de un programa de ejecución secuencial en Python que, una vez está levantada la cloud, inicia todo el sistema de ejecución. Los parámetros de llamada de este programa indica el modo en el que se quiere inicial el sistema, que son USB, modo WiFi a través de Logcat IP y por ultimo otro modo WiFi con la estructura cliente-servidor.

"python device_launcher.py <opciones>"

1. "python device_launcher.py" para modo usb.
2. "python device_launcher.py 1 <terminalIp>" para capturar los eventos a través de logcat ip.
3. "python device_launcher.py 2 <serverIp>" para ejecutar el archivo listener que se encuentra en la carpeta /utils del sistema, y el archivo /andro_talker que se envía al terminal físico. La ip se corresponde con la propia del equipo/servidor que aloja todo el sistema de ejecución. En vez de ip se puede poner el nombre de máquina ya que realiza una búsqueda DNS.

Cada una de estas opciones ejecuta unos mismos pasos, que son:

- Desinstalar la aplicación: es.uc3m.cosec.captureevents
- Desinstalar la aplicación: es.uc3m.cosec.replicator
- Forzar la reinstalación de la aplicación: es.uc3m.cosec.captureevents
- Forzar la reinstalación de la aplicación: es.uc3m.cosec.replicator
- Comenzar la actividad que comienza el servicio de captura de eventos propios.
- Limpiar el registro con: logcat –c
- A través de device_monitoring_X.sh comenzar a capturar eventos con logcat, que van a comunicarse con una tubería con el programa: “device2clone.py”

- **device2clone.py**

Este programa lee de la entrada estandar cada uno de los eventos enviados por device_monitoring_X.sh. Cada evento se decodifica para eliminar caracteres que puedan producir error, y se hacen un par de comprobaciones de validación de entrada. Estas comprobaciones son que el evento tenga un tipo de evento definido (nombre de evento no vacio) y que esté dentro de la lista blanca declarada como variable:

```
EVENTENABLEDLIST = ["ActivityManager", "PackageManager",
"AirplaneModeEnabler", "CAPTUREVENTS", "MmsSmsProvider",
"GpsLocationProvider", "ConnectivityService", "PowerManagerService"];
```

Si cumple estas condiciones entonces es un evento que va a actuar como input del sistema, si no se descarta.

Para continuar con la ejecución se crea un objeto de tipo Event, y se ejecuta cloneEvent.clone_run().

- **LogcatEvent.py**

Este programa posee la funcionalidad principal del sistema. Cuando se crea un objeto tipo Event se comprueba que el tipo de evento pasado por parámetro existe en alguna de las clases implementadas a partir de la template. Es decir, para poder crear un Evento se comprueba que el nombre del evento, por ejemplo: “PackageManager” coincida con el nombre de alguna clase implementada en LogcatEvent.py, como es el caso:

```
class PackageManager(RunnerEvent):
# Inputs: D/PackageManager(  527): Removing package com.estrongs.android.pop
# Inputs: I/PackageManager( 1130): Removing non-system package:android.pruebas.bateria2
def clone_run(self):
   if "Removing package" in self.elements[1] :
      paquete = self.elements[1].strip().split(" ")[2]
      call(['adb', "-s", self.deviceId, "uninstall", paquete], stderr=PIPE)
   elif "Removing" in self.elements[1] and "package" in self.elements[1] and ":" in self.elements[1]:
      paquete = self.elements[1].strip().split(":")[1]
      call(['adb', "-s", self.deviceId, "uninstall", paquete], stderr=PIPE)
```

En concreto esta clase, al ejecutar el método clone_run(), desinstala la aplicación objetivo en la cloud. La aplicación objetivo es *parseada* según los elementos del objeto Evento, que coinciden con el mensaje del evento raw. La implementación se puede ver en el método: _parse_() de la clase Event.

Así se procede para cada uno de los eventos gestionados por el sistema.

Como se observa, la acción realizada es una llamada al programa adb, que tiene como parámetro uninstall y el paquete *parseado* del evento raw. Cada evento tiene su acción o acciones que se implementan de acuerdo al análisis realizado, bien utilizando herramientas externas como en este caso adb, o bien generando un *Broadcast* de comunicación con la aplicación instalada en la cloud, que se explica en el apartado siguiente.

Para finalizar vamos a ver un ejemplo de acción con Telnet. Suponemos que llega al sistema un evento de tipo CAPTUREVENTS, y en concreto uno de cambio en la localización del terminal físico:

```
CAPTUREVENTS(  1812):  TAGLOCATION  |  Lat:numberLat  |
```

La acción a realizar, según el análisis, es ejecutar a través de Telnet un comando que modifica la localización del dispositivo virtual. Se *parsea* el mensaje del evento obteniendo las variables Lat y Long.

```
lat = resto[1].split(":")[1].strip()
lon = resto[2].split(":")[1].strip()
objeto = Comando("geo fix "+lon+" "+lat)
res = objeto.execTelnet()
```

7.4. Implementación de aplicación Replicator

Esta aplicación está formada únicamente por un *BroadcastReceiver*. La idea es la misma que la vista en la clase Receiver de la aplicación CaptureEvents. En este caso con un switch se espera un código, que representa un tipo de acción determinada, mientras que por parámetros (extras) se envía información detallada sobre qué acción realizar.

Un ejemplo de *Broadcast* emitido desde LogcatEvent.py es por ejemplo:

```
adb -e shell am broadcast -a es.uc3m.cosec.replicator.REPLICATE --ei
```

En concreto esta acción posee parámetros, únicamente indica en el código 4, que se ha producido en el terminal físico un cambio de fondo de pantalla.

La acción a realizar desde la aplicación replicator por tanto es:

```
// code = 4 | Wallpaper changed -> set default included
// adb -e shell am broadcast -a es.uc3m.cosec.replicator.REPLICATE --ei code 4
case 4:
        WallpaperManager myWallpaperManager =
WallpaperManager.getInstance(context);
  try {
    myWallpaperManager.setResource(R.drawable.andro);
  } catch (Exception e) {
        Log.d(TAG, " Exception: "+e.toString());
  }
```

La imagen que se pone de fondo de pantalla en la cloud es una incluida por defecto en la aplicación. No se ha encontrado manera de determinar qué imagen se cambia en el terminal físico.

7.5. Scripts y utilidades

Por último relativo a la implementación se comentan un par de scripts que tienen una funcionalidad muy importante en el sistema.

- **startupcloud.sh**

Este script se debe ejecutar en primer lugar antes de device_launcher.py, ya que levanta la cloud correspondiente. Si no se pasa parámetro levanta por defecto: clonecloud. Si se desea utilizar otro Android Virtual Device como cloud, basta con pasar el nombre como parámetro.

En este script se utilizan las imágenes del sistema elaboradas para Droidbox, por lo que se carga en la cloud el sistema Droidbox que permite la monitorización de aplicaciones malware.

- **killingAdb.py**

Este script se ha realizado para limpiar los procesos adb del servidor, ya que suelen muy comúnmente quedarse colgados produciendo errores de comunicación en el sistema, e incluso a veces la detención del mismo.

Básicamente se realiza un filtrado de los procesos del sistema con: "ps -ef | grep adb | awk {print$2}", que devuelve una lista con todos los identificadores de los procesos adb existentes. En un for se matan uno a uno para limpiar siguiente ejecución del sistema.

Sección 8

EVALUACIÓN E IMPLANTACIÓN

8.1. Evaluación y Pruebas

Por naturaleza de este sistema vamos a distinguir dos tipos de pruebas, las primeras relacionadas con los requisitos, y las segundas relacionadas con el testeo de los propios eventos de replicación analizados para su implementación en el sistema.

Las pruebas del primer grupo son de concepto y justificación de como el sistema cumple con los requisitos. En el segundo grupo se realizan pruebas de caja negra, comprobando la correcta replicación de los principales eventos implementados.

El entorno de todas las pruebas realizadas consta de un terminal Android con firmware de stock y versión 2.3.3, en concreto un Samsung modelo Galaxy S. En el lado de la nube se ha levantado un dispositivo virtual Android con versión 2.3.3 (API 10).

8.1.1. Testeo de requisitos de Software

La funcionalidad y cumplimiento de varios requisitos se puede verificar con una única prueba en muchos casos. Por tanto este primer apartado establece por conjunto de requisitos de SW relevante una prueba que valide el correcto funcionamiento del sistema. Se han realizado las siguientes pruebas unitarias.

Código de prueba: PSW-01	
Requisito de SW asociado	RS-01; RS-02; RS-11; RS-12; RS-13; RS-14; RS-16
Descripción	Comprobar el comportamiento del terminal físico a través de eventos de sistema y de eventos propios
Localización de la funcionalidad	device_launcher.py, Aplicación CaptureEvents
Resultado.	En cada una de las tres opciones de ejecución de device_launcher.py se llama a un script que realiza la llamada a logcat para obtener los eventos del sistema así como los generados por la aplicación.

Código de prueba: PSW-02	
Requisito de SW asociado	RS-03, RS-04, RS-09; RS-11; RS-12; RS-13;
Descripción	Los eventos capturados actúan como input del sistema, son tipificado y se implementan las distintas acciones que pueda tener
Localización de la funcionalidad	device_monitoring.sh, device2clone.py, LogcatEvent.py
Resultado.	device_monitoring.sh captura eventos con Logcat y son enviados como input a device2clone.py para su decodificación, se crea un Evento según el tipo cuya implementación evalúa la/s acciones a realizar

Código de prueba: PSW-03	
Requisito de SW asociado	RS-07; RS-12; RS-13; RS-15
Descripción	Los eventos son replicados en la cloud
Localización de la funcionalidad	LogcatEvent.py
Resultado.	El objetivo de las acciones realizadas en cada clase de LogcatEvent.py viene determinado por el idevice (identificador de dispositivo) objetivo, que es el código de la cloud: emulator-5554

Código de prueba: PSW-04	
Requisito de SW asociado	RS-08, RS-10; RS-17; RS-18
Descripción	Los eventos que no se pueden replicar de manera externa son replicados en la cloud
Localización de la funcionalidad	LogcatEvent.py , Aplicación Replicator
Resultado.	Los *Broadcast* emitidos por LogcatEvent.py son capturados por la aplicación Replicator que ejecuta la acción deseada

Código de prueba: PSW-05	
Requisito de SW asociado	RS-11; RS-19, RS-20
Descripción	Los eventos se capturan por USB y por WiFi
Localización de la funcionalidad	device_launcher.py
Resultado.	device_launcher.py implementa tres opciones de ejecución incluyendo una USB y dos WiFi

Código de prueba: PSW-06	
Requisito de SW asociado	RS-12; RS-21
Descripción	Se validan los datos de entrada
Localización de la funcionalidad	device2clone.py
Resultado.	Se decodifica cada evento para evitar caracteres no soportados.

Código de prueba: PSW-07	
Requisito de SW asociado	RS-12; RS-22
Descripción	Se tratan las posibles excepciones
Localización de la funcionalidad	device2clone.py, LogcatEvent.py
Resultado.	En los posibles casos que puedan generar excepciones por errores de tipado de datos se implementan estructuras de try/catch

Código de prueba: PSW-08	
Requisito de SW asociado	RS-23
Descripción	Se aseguran tiempos de respuesta inferiores a 10 segundos
Localización de la funcionalidad	N/A
Resultado.	Se suprime la funcionalidad de captura de eventos de sistema mitigando el riesgo que produce la gran demora causada por el inmenso número de eventos de este tipo

8.1.2. Testeo de replicación de acciones

Todas las acciones incluidas en el anexo 2 se han testeado durante la fase de análisis, diseño e implementación para comprobar que efectivamente funciona todo dentro del alcance y del entorno de prueba establecido.

A continuación se detallan las pruebas realizadas, se divide entre las acciones relacionadas con el malware (Prueba Acción Malware), así como las pruebas relacionadas con las acciones de usuario (Prueba Acción Usuario).

Código de prueba: PAM-01	
Requisito de SW asociado	RS-05
Descripción	Cambio de estado de batería
Localización de la funcionalidad	LogcatEvent.py, clase CAPTUREVENTS
Resultado.	Se comprueba la correcta replicación de la acción.

Código de prueba: PAM-02	
Requisito de SW asociado	RS-05
Descripción	Gestión de llamadas
Localización de la funcionalidad	LogcatEvent.py, clase CAPTUREVENTS. Aplicación Replicator clase Receiver
Resultado.	Se comprueba la correcta replicación de las acciones: realizar llamada, recibir llamada, colgar llamada y cancelar llamada.

Código de prueba: PAM-03	
Requisito de SW asociado	RS-05
Descripción	Gestión de Red
Localización de la funcionalidad	LogcatEvent.py, clases ConectivityService y AirplaneModeEnabler. Aplicación Replicator clase Receiver
Resultado.	Se comprueba la correcta replicación de las acciones: habilitar y deshabilitar modo avión, así como habilitar y deshabilitar red de datos.

Código de prueba: PAM-04	
Requisito de SW asociado	RS-05
Descripción	GPS
Localización de la funcionalidad	LogcatEvent.py, clases GpsLocationProvider y CAPTUREVENTS. Aplicación Replicator clase Receiver
Resultado.	Se comprueba que las acciones de apagar y encender GPS se replican correctamente, además de replicarse el cambio de localización.

Código de prueba: PAM-05	
Requisito de SW asociado	RS-05
Descripción	Gestión de paquetes
Localización de la funcionalidad	LogcatEvent.py, clases PackageManager y ActivityManager
Resultado.	Se comprueba que se replican correctamente las acciones de: iniciar una aplicación instalada, instalar una aplicación y desinstalar una aplicación.

Código de prueba: PAU-01	
Requisito de SW asociado	RS-06
Descripción	Ajuste de hora/fecha
Localización de la funcionalidad	LogcatEvent.py, clase CAPTUREVENTS.
Resultado.	Se comprueba el correcto cambio de hora y fecha en la nube.

Código de prueba: PAU-02	
Requisito de SW asociado	RS-06
Descripción	Modificación de fondo de pantalla
Localización de la funcionalidad	LogcatEvent.py, clase CAPTUREVENTS. Aplicación Replicator clase Receiver
Resultado.	Se comprueba la correcta replicación del evento cambio de fondo de pantalla.

Código de prueba: PAU-03	
Requisito de SW asociado	RS-06
Descripción	Modificación de volumen
Localización de la funcionalidad	LogcatEvent.py, clase CAPTUREVENTS. Aplicación Replicator clase Receiver
Resultado.	Se comprueba la correcta replicación de los eventos: subir volumen, bajar volumen, activar modo silencio, activar modo vibrador y activar modo normal.

En la siguiente hoja se muestra la matriz de trazabilidad que enlaza las pruebas realizadas junto con los requisitos de software definidos en el apartado de analisis.

Matriz de trazabilidad de Requisitos de SW - Pruebas Realizadas

		Requisitos de Software																					
		RS-01	RS-02	RS-03	RS-04	RS-05	RS-06	RS-07	RS-08	RS-09	RS-10	RS-11	RS-12	RS-13	RS-14	RS-15	RS-16	RS-17	RS-18	RS-19	RS-20	RS-21	RS-22	RS-23
Prueba Realizada	PSW-01	X	X									X	X	X	X		X							
	PSW-02			X	X					X		X	X	X										
	PSW-03							X					X	X		X								
	PSW-04								X		X							X	X					
	PSW-05											X								X	X			
	PSW-06												X									X		
	PSW-07												X										X	
	PSW-08																							X
	PAM-01					X																		
	PAM-02					X																		
	PAM-03					X																		
	PAM-04					X																		
	PAM-05					X																		
	PAU-01						X																	
	PAU-02						X																	
	PAU-03						X																	

Tabla - 12 – Matriz de Trazabilidad. Pruebas – Req. Sw

8.2. Implantación y Mantenimiento

Para poner el sistema en producción se requiere de una serie de pasos y elementos que condicionan en gran medida el uso del mismo. Lo más importante de todo es poseer un terminal Android cuyas acciones de usuario alimentan el sistema. En caso de no poseer terminal Android, se puede adaptar el sistema a la replicación de acciones de un dispositivo virtual.

El entorno de trabajo está limitado a una shell de Unix para poder ejecutar los scripts de comunicación. También se requiere de las librerías de Python para poder ejecutar los principales módulos del sistema. Por último se necesita tener instalado el Android SDK, así como un dispositivo virtual creado que actúa de cloud.

La implantación se realizará en un repositorio online con acceso público para que la comunidad de desarrolladores evolucionen el proyecto y creen un ciclo de vida que mantenga el sistema en constante progreso.

Un aspecto muy importante que se tiene en cuenta es la licencia. El sistema publicado se considera software libre, por lo que se publicará bajo licencia de software libre. En concreto se utilizará la Licencia Pública General de GNU (GPL de GNU).

El mantenimiento se realizará atendiendo los comentarios y el feedback que puedan proporcionar los interesados en este sistema que accedan al repositorio.

A continuación se muestra un manual de usuario para poner el sistema en funcionamiento.

OVERVIEW

This System replicates events from a physical android device to an AVD.

There are 2 types of events:

1. System. This kind of events is generated by apps or the system. The action must be programmed just in coseccloneclоud/device2clone/LogcatEvent.py as a new class-template. The last aim is not to use such events so as to not depend on the Android version of the device.

2. CapturEvents. The events are generated by the customizable Apk which source code is located in the CaptureEvents/ project. This app generates CAPTUREVENTS events type by listening the behavior of the terminal. There are 4 types of functions implemented so far:

a. BroadcastReceiver. Gives information about what happen in the device.

b. ContentObserver. Related to ContentProvider changes.

c. EventListener. Some types of EventListener you can register.

d. RegisteredBroadcast. A few specific Broadcast not possible to register in AndroidManifest.

Some actions just cant be replicated in a external way (telnet, adb, monkey, native utils...), so you can find in Replicator/ project a BroadcastReceiver class with a single and custom broadcastreceiver. This class listen to the code sended as extra, and execute the lines implemented for that specific code.

This way you can log whatever you want in a base logcat event, that is replicated by the system with the correct code in the CAPTUREVENTS class-template.

MAKE IT WORKING

Sección 9

CONCLUSIONES

9.1. Conclusión y Líneas Futuras

El proyecto realizado es un sistema útil y pragmático. El resultado es muy positivo, el sistema se ha convertido en una herramienta con un ciclo de vida seguro, por lo que este Trabajo de Fin de Grado no supone un trámite o asignatura más para obtener un título. También se valoran los conocimientos adquiridos y el desarrollo de todo el proyecto.

Partiendo de los artículos indicados en el estado del arte, el sistema desarrollado puede servir como base para algunos de ellos, como por ejemplo CloneCloud o ParanoidAndroid. El sistema por tanto, sirve como base para que otras personas sigan investigando la misma u otras líneas similares de trabajo.

Por otro lado, la realización de un proyecto más técnico y práctico que otro de documentación e innovación teórica fuerza al aprendizaje autónomo, como ha sido el caso del funcionamiento de Android. En concreto a un nivel más bajo del que se puede aprender en el desarrollo más extendido como es el de juegos o aplicaciones orientadas al ocio. También es muy motivador realizar proyectos orientados a la actividad profesional que uno desea realizar, como es la seguridad y en este caso el ámbito de la seguridad en dispositivos móviles.

A pesar de que el proyecto está muy acotado por el software utilizado, la evolución del proyecto ha sido dinámica e innovadora. El fuerte peso del análisis en el proyecto ha obligado a realizar una parte de innovación que es la inclusión de las aplicaciones Android para poder abstraer casi por completo la versión de Android a utilizar en el terminal físico.

El resultado es bastante prometedor, se ha comprobado el funcionamiento de un bueno número de acciones y su replicación, y el sistema no es reticente a la creación e incorporación de nueva funcionalidad. Con un análisis de las acciones nuevas que se desean incorporar al sistema se puede determinar si los eventos se capturan directamente por log de sistema o si es necesario añadir funcionalidad a la aplicación CaptureEvents. Como posible mejora o alternativa a la captura de eventos se podría utilizar también la herramienta NXLOG[7] que se integra con Android y otros sistemas para recolectar y enviar eventos.

De igual modo se debe proceder con la replicación. Las herramientas externas que permiten la interacción con la cloud son bastante limitadas ya que no generan un ciclo de vida normal en los procesos internos del Sistema Android. Esto puede ser beneficioso para la evolución del sistema en caso de que se interesen perfiles técnicos con conocimientos de Android.

Como próximos pasos adicionales, decir que el entorno cloud todavía es mejorable. Al no tratarse de equipos en producción como los terminales físicos, se podría investigar las ventajas de realizar aplicaciones de sistema, ganar permiso de administrador (*root*) en el dispositivo virtual, etc. Esto supondría una funcionalidad mucho mayor en el control y monitorización del cloud. Sería también muy relevante realizar estudios sobre el consumo de energía necesaria para replicar la información.

[7] NXLOG

Sección 10

ANEXOS

10.1. Anexo 1 - Evento - Acción inicial

Al ser excesivamente grande la tabla en excel para que se visualice correctamente, se adjunta dentro del apartado el archivo excel que lo contiene.

10.2. Anexo 2 - Evento - Acción final

Además de las capturas, se incluye el propio documento para poder consultar sobre el Excel la tabla:

10.3. Anexo 3 - Referencias

[1] G. Suarez-Tangil, "Evolution, Detection and Analysis of Malware for Smart Devices," 2013.

[2] J. N. Inc., "2011 Mobile Threats Report," Juniper Networks Inc., February 2012.

[3] J. N. Inc., "2012 Mobile Threats Report," Juniper Networks Inc., March 2013.

[4] G. Portokalidis, P. Homburg, K. Anagnostakis and H. Bos, "Paranoid Android: Zero-Day Protection for Smartphones Using the Cloud".

[5] T. Labs, "2014 1Q Security Roundup," 2014.

[6] M. S. Inc., "Marble Labs Mobile Threat Report, June 2014," 2014.

[7] B. O. d. Estado, "Ley Orgánica de Protección de Datos de carácter personal," 1999.

[8] C. C. Nacional, "Seguridad en Dispositivos Móviles".

[9] C. C. Nacional, "Seguridad en Dispositivos Móviles: Android".

[10] R. Fredler, J. Schütte and M. Kulicke, "On the Effectiveness of Malware," Abril 2013.

[11] V. Rastogi, Y. Chen and X. Jiang, "Evaluating Android Anti-malware against," March 2013.

[12] G. Portokalidis, P. Homburg, K. Anagnostakis and H. Bos, "Paranoid Android: Versatile Protection For Smartphones".

[13] http://elinux.org, "Android Logging System," [Online]. Available: http://elinux.org/Android_Logging_System.

[14] Yajin Zhou and Xuxian Jiang, "Analysis of AnserverBot", September 25, 2011.

[15] J. N. Inc., "Maliciuos Mobile Threats Report 2010/2011", Juniper Networks Inc., May 2011

[16] http://contagiodump.blogspot.com.es, "Contagio Mobile" [Online]. Available: http://contagiodump.blogspot.com.es/

[17] http://www.alienvault.com, "Analysis of Trojan-SMS.AndroidOS.FakePlayer.a" [Online]. Available: http://www.alienvault.com/open-threat-exchange/blog/analysis-of-trojan-smsandroidosfakeplayera

[18] F-Secure, "Mobile Threat Report Q4 2012", F-Secure Labs, March 2013

[19] http://www.forbes.com, "To Hide Android Malware Apps From Google's 'Bouncer', Hackers Learn Its Name, Friends, And Habits" [Online]. Available: http://www.forbes.com/sites/andygreenberg/2012/06/04/to-hide-android-malware-from-googles-bouncer-hackers-learn-its-name-friends-and-habits/

[20] Jon Oberheide and Charlie Miller, "DISSECTING THE ANDROID BOUNCER" [Online]. Available: https://jon.oberheide.org/files/summercon12-bouncer.pdf

[21] Byung-Gon Chun, S unghwan Ihm and P etros Maniatis, "CloneCloud: Elastic Execution between Mobile Device and Cloud" 2011.

[22] Sokol Kosta, Andrius Aucinas, Pan Hui, Richard Mortier and Xinwen Zhang, " ThinkAir: Dynamic resource allocation and parallel execution in cloud for mobile code offloading" 2012.

10.4. Anexo 4 - Diagramas de Gantt

Diagrama de Gantt Inicial:

Diagrama de Gantt de replanificación: